MS 87-8 SERIES 2 BOX 3 FF 15

HAL/S-FC COMPILER SYSTEM SPECIFICATION

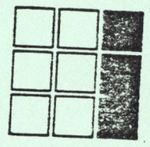
IR-95-6

13 January 1977

INTERMETRICS APPROVAL

Daniel J. Lickly
Head, HAL Language Compiler Department

Dr. F. H. Martin
Shuttle Program Manager



INTERMETRICS

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

FOREWORD

This document was prepared for the Johnson Space Center, Houston, Texas, under contract NAS 9-13864.

The HAL/S-FC Detail System Specification was prepared by the staff of Intermetrics, Inc., under the direction of Mr. Ronald E. Kole.

Table of Contents

		Page
1.0	INTRODUCTION	1-1
1.1	Scope of Document	1-1
1.2	Outline of the Document	1-1
1.3	Status of Document	1-2
2.0	PHASE I - SYNTAX ANALYSIS	2-1
2.1	Primary Source Input	2-2
2.2	Secondary Source Input - The INCLUDE System	2-3
2.3	ACCESS Rights Implementation	2-4
2.4	Compiler Directive Parsing	2-6
2.5	Template Checking and Generation	2-6
2.6	Listing Generation	2-8
2.6.1	Primary Formatted Listing	2-8
2.6.2	Error Messages	2-9
2.6.3	Block Summaries	2-10
2.6.4	Compilation Layout Summary	2-11
2.6.5	Symbol & Cross Reference Table Listing	2-11
2.6.6	Built-in Function Cross Reference	2-11
2.6.7	Replace Macro Text	2-12
2.6.8	Unformatted Source Listing	2-12
2.7	Symbol Table Generation	2-12
2.8	Statement Table Generation	2-13
2.9	Literal Table Generation	2-13
2.10	HALMAT Creation	2-14
2.11	The Optimizer	2-14
3.0	PHASE II - CODE GENERATION	3-1
3.1	Code Generation	3-1
3.1.1	Bases and Conventions	3-1
3.1.2	Integer and Scalar Operations	3-18
3.1.3	Bit String Operations	3-24
3.1.4	Character String Operations	3-29
3.1.5	Vector Matrix Operations	3-32
3.1.6	Structure Operations	3-46
3.1.7	Indexing and Arrayed Statements	3-48
3.1.8	PROCEDURE/FUNCTION Calls	3-52
3.1.9	Block Definition	3-54
3.1.10	Flow of Control Statements	3-55
3.1.11	Built-In Functions	3-64
3.1.12	Real Time Statements	3-70
3.1.13	I/O Statements	3-74
3.1.14	NAME Operations	3-80
3.1.15	%MACROS	3-81
3.1.16	NONHAL References	3-81
3.1.17	Block Move Algorithm	3-82

Table of Contents (Continued)

		Page
3.2	Object Code Naming Conventions	3-83
3.3	Printed Data From Phase II	3-84
3.4	Symbol Table Augmentation	3-84
3.5	Statement Table Augmentation	3-85
4.0	PHASE III - SIMULATION DATA FILE GENERATION	4-1
4.1	SDF Generation	4-1
4.1.1	Overall SDF Design	4-1
4.2	Phase III Printed Data	4-3
5.0	RUN TIME LIBRARY	5-1
5.1	Introduction	5-1
5.2	Basics and Conventions	5-1
5.2.1	Origin and Format	5-1
5.2.2	Purpose	5-1
5.2.3	Intrinsics and Procedure Routines	5-1
5.2.4	Register Conventions in Run Time Library Routines	5-2
5.2.5	Referencing Conventions	5-2
5.2.6	Coding Structure	5-2
5.2.7	The Macro Library	5-2
5.3	Library Routine Descriptions	5-10
5.3.1	Arithmetic Routine Descriptions	5-11
5.3.2	Algebraic Routine Description	5-12
5.3.3	Vector/Matrix Routine Descriptions	5-31
5.3.4	Character Routine Descriptions	5-52
5.3.5	Array Function Routine Descriptions	5-71
5.3.6	Miscellaneous Routine Descriptions	5-135
5.3.7	REMOTE Routine Descriptions	5-243
6.0	SYSTEM INTERFACES	5-268
6.1	Internal System Interfaces	5-285
6.1.1	Macro Instructions	5-332
6.1.2	Dynamic Invocation of the Compiler	6-1
6.1.3	OS/360 Access Methods	6-1
		6-1
		6-2
		6-2

Table of Contents (Continued)

		Page
6.2	User or External System Interfaces	6-3
6.2.1	User-defined Options	6-3
6.2.2	Job Control Language Specification	6-3

Appendices:

Appendix A:	Compile-Time JCL Options	A-1
Appendix B:	Compiler Directives	B-1
Appendix C:	Error Classifications	C-1

1.0 INTRODUCTION

1.1 Scope of Document

This document specifies the informational interfaces within the HAL/S-FC compiler, and between the compiler and the external environment. An overall description of the compiler, and the hardware and software compatibility requirements between compiler and environment are detailed in the HAL/S-FC Compiler Functional Specification[1]. Familiarization with the Functional Specification is presumed throughout this document.

This Compiler System Specification is for the HAL/S-FC compiler and its associated run time facilities which implement the full HAL/S language[2]. The HAL/S-FC compiler is designed to operate "stand-alone" on any compatible IBM 360/370 computer and within the Software Development Laboratory (SDL) at NASA/JSC, Houston, Texas.

1.2 Outline of the Document

The HAL/S-FC compiler system consists of:

1) a five phase language processor (compiler) which produces object modules compatible with AP-101 Space Shuttle Support Software and a set of simulation tables to aid in run time verification.

2) a comprehensive run-time library which provides an extensive set of mathematical, conversion, and language support routines.

The organization of this document is based upon the organization of the compiler system. Each part of the system is considered as a separate entity with its own specific function and interfaces to other parts. Hence, there are four sections which cover the parts of the system as follows:

---

1 HAL/S-FC Compiler System Functional Specification, 24 July 1974, IR #59-4.

2 HAL/S Language Specification, 14 November 1975, IR #61-7.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Section 2 - describes Phase I and the syntax analysis phase of the compiler.

Section 3 - describes Phase II and the code generation phase and specifies in detail the code patterns for specific HAL/S constructs.

Section 4 - describes Phase III and the operation of the Simulation Data File generator.

Section 5 - describes the Runtime Library and the concepts used in the library and also gives specific information about each library routine including size, speed, and algorithm.

In addition to this part-by-part documentation, the compiler system, taken as a whole, exhibits properties and interfaces which are not specific to any one of the pieces. General information about such topics as the compiler's operating environment and user-written interfaces to emitted object code are contained in Section 6. Several Appendices are included which deal with tabular data used in the compiler system.

1.3 Status of Document

This document, plus the HAL/S-FC Compiler System Functional Specification comprise the complete HAL/S-FC Compiler System Specificaiton. This publication is a specification for Release 10.0 of the HAL/S-FC compiler system.

The HAL/S-FC compiler inherits some of its operational features from the HAL/S-360 compiler system for which a similar Specification exists. In addition, many features of the HAL/S-FC system are under control of Interface Control Documents which are subject to update. When appropriate within this document, references are made to these companion documents as sources of supplementary material and in some cases as primary sources of detailed information.

The following list of documents represents the set of additional documents which reflect design and control of the HAL/S-FC compiler system:

- HAL/S-FC Compiler System Functional Specification, IR #59-4, 24 July 1974, by Intermetrics, Inc.
- Interface Control Document: HAL/FCOS, Revision 5 Published by IBM Federal Systems Division, Houston, Texas.
- Interface Control Document: HAL/SDL, Revision 6, Published by IBM Federal Systems Division, Houston, Texas.
- HAL/S-360 Compiler System Specification, IR #60-6 by Intermetrics, Inc.
- HAL/S Language Specification, IR #61-8, Published by Intermetrics, Inc.

2.0 PHASE I - SYNTAX ANALYSIS

The Syntax Analysis Phase performs syntactic and semantic analysis of the user's HAL/S source programs. It performs all functions necessary to allow an independent Phase II program to generate code for the target computer. The basic design of the HAL/S system includes use of a single Phase I for a variety of target machine Phase II's. Thus, the Phase I used by the HAL/S-FC compiler is the same one used in the HAL/S-360 compiler. In this section on Phase I, data which is supplied in detail in the HAL/S-360 Compiler System Specification is not repeated. Instead, reference is made to the proper section of that document.

This section deals with the following Phase I functions:

- Primary Source Input
- Secondary Source Input
- ACCESS System Implementation
- Compiler Directives
- Template Checking and Generation
- Printed Data
- Symbol Table Creation
- Statement Table Generation
- Literal Table Generation
- HALMAT Creation
- The Optimizer

Primary Source Input

Phase I accepts primary source input in the form of fixed length logical records. This input must be defined by the SYSIN DD statement in the JCL invoking the compiler. The first byte of each record is used to define the type of the record as follows:

M - main line
E - exponent line
S - subscript line
D - compiler directive
C - comment

(Other letters can be used if modified via the "CARDTYPE" compiler option to a legal type.)

For stand-alone operation the source records are 80 bytes in length and may contain data in columns 2-80. Optionally, the user may designate, via the "SRN" compiler option, that the source scanning is to stop at position 72 and also that positions 73-78 are to be printed on the listing as "Statement Reference Numbers".

When operating in the SDL environment, indicated by use of the "SDL" compiler option, the source records must still be all the same length but that length may be from 80 to 132 characters. When in the SDL mode, the compiler accepts source data from record positions 2 through 72. In addition, when the records are of sufficient length, the following fields are recognized and printed on the primary source listing:

- Record Sequence Number - positions 73 through 78;
- Record Revision Indicator - positions 79 and 80;
- Change Authorization Field - positions 81 - 88.

ATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Portions of records beyond position 88 are ignored.

The compiler's primary input may optionally be in a compressed source format as defined in the HAL/SDL ICD. No special notification of use of compressed source is needed. Phase I determines the type of input by examining the first record. Catenated datasets defined as primary compiler input must all be either in compressed or non-compressed format for one invocation of the compiler.

2.2 Secondary Source Input - The INCLUDE System

The user may direct the compiler to an alternate input source by use of an INCLUDE compiler directive in the primary input. The exact form of the INCLUDE directive may be found in Appendix B.

The INCLUDE directive defines a member name in a partitioned dataset. Phase I uses a FIND macro to locate the member on the INCLUDE DD card. If the FIND is unsuccessful, an identical FIND is issued for the OUTPUT6 DD card. A member, when located, is read to its end by the compiler. The records are processed identically to primary (SYSIN) input with the exception that further INCLUDE directives within INCLUDE'd source are not allowed. The same source margins are applied to the INCLUDE'd source as are applied to the primary input. In addition, the compiler prints a line in the primary source listing indicating the catenation sequence number of the DD card on which the member was found and the RVL field from the PDS directory entry for the member. The RVL field is the first 2 bytes of user data after any TTRN's.

The individual members which are INCLUDE'd may be in either compressed or uncompressed format, independent of whether the primary input was compressed. The form of each INCLUDE'd member is determined by the compiler from the first record read.

Partitioned datasets may be catenated together in the JCL to form the INCLUDE DD sequence, but such datasets must have identical DCB attributes.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.3 ACCESS Rights Implementation

The HAL/S language allows managerial restrictions to be placed upon the usage of user-defined variables and external routines. The existence of such a restriction is indicated by the use of the ACCESS attribute as described in the HAL/S Language Specification. The manner in which the restrictions are enforced in the HAL/S-FC compiler system is described below.

Any variable in a COMPOOL template or any external routine to which the ACCESS attribute has been applied is considered to be restricted for the compilation unit which is being compiled. The restriction is slightly different for variables than for blocks:

a) Variables with the ACCESS attribute may not have their values changed.

b) Block names may not be used at all.

These restrictions may be selectively overridden for individual variable and block names. The selection of which ACCESS controlled names are to be made available to the unit being compiled is performed by processing an external dataset. The external dataset is known as the Program Access File (PAF). The PAF must have partitioned organization and is specified by the following JCL:

```
//HAL.ACCESS DD DSN=<PAF name>, <other parameters>
```

where the <PAF name> is the dataset name of the PAF without any member specification.

Each member of the PAF contains the information about ACCESS controlled names which are to be made available to one unit of compilation. The member name is defined by a Program Identification Name (PIN). The PIN is specified to the HAL/S-FC compiler by using the PROGRAM compiler directive in the primary input stream:

```
col 1
 D
     PROGRAM ID = <id>
```

The <id> field of the directive is a 1 to 8 character identifying name which is used to select the member of the PAF to be processed for the current compilation's ACCESS information. The appearance of the PROGRAM directive in the compiler's input stream causes immediate processing of the PAF member specified.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

The format of an individual PAF member is described below.

a) Column 1 of each record is ignored except when column 1 contains the character "C", in which case the entire record is ignored.

b) The portion of each record which is processed is the same portion which is processed in the primary compiler input (SYSIN).

c) COMPOOL elements which are to be made available to the compilation are specified as:

```
<COMPOOL-name> (<var-name>, <var-name>, ... <var-name>)

 or

<COMPOOL-name> ($ALL)
```

The first format specifies access to individual variables within the named COMPOOL. The second format specifies access to all variables within the named COMPOOL.

d) Access to external block names is specified as:

```
$BLOCK(<ext-name>, <ext-name>, ... <ext-name>)
```

e) Blanks are allowed anywhere in the record except that names may not be broken by a blank.

f) Either of the constructions (c) or (d), above, may span more than one record.

g) The name of the particular COMPOOL in the form (c) above may appear more than once; i.e. the variables in a particular COMPOOL do not have to be specified at one time . Similarly, the form $BLOCK may appear more than once.

Some validity checking is performed by the compiler while processing the PAF member. Warnings are issued for the following conditions:

1) A syntax error on a PAF record - the bad record is printed.

2) Names mentioned in the PAF are not defined.

3) Elements of $BLOCK in the PAF are not defined.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

4) Requests for names which are not ACCESS protected.

5) Variables found, but not within the COMPOOL specified.

6) Names used in the context of a COMPOOL-name which are not COMPOOLs.

If, at the time the PROGRAM directive is encountered, there have been no ACCESS-controlled variables declared, the PAF is not opened. If a user does not require access to any, the PROGRAM directive and associated PAF members may be omitted.

2.4 Compiler Directive Parsing

When an input record is found which contains a "D" in column one, Phase I scans the remainder of the card for a valid compiler directive. A list of legal compiler directives and their function is listed in Appendix B.

Directive processing is done independently of HAL/S source language parsing, i.e. words used on Directive cards are not necessary HAL/S language keywords. Similarly, HAL/S language keywords are not recognized as such on Directive cards.

2.5 Template Checking and Generation

Phase I assumes the task of source template verification and generation. Every compilation unit in the HAL/S-FC system has a source template. When the block header for a unit of compiler is encountered, Phase I begins to construct the source template for that unit as follows.

The member name for the template being created is determined. This is done by taking the "characteristic name" for the unit and preceeding it by the characters '@@'. The characteristic name for any unit is created by taking the block name, removing any underscore characters, and then padding or truncating the result to 6 characters. An attempt is made to locate a member of this name on either the INCLUDE or OUTPUT6 DD cards. If such a member is found, the contents of the member are compared with an internal, temporary template created as the compilation proceeds. If the existing template and the internal one

agree, a template update is not required, and the existing template remains intact. If the templates do not agree, the internal template is written to the OUTPUT6 DD card and STOW'ed with the current member name. If the initial search for an existing template fails, the generated template is automatically written and STOW'ed on the OUTPUT6 DD card. The PDS directory entry for a template member is created with two bytes of user data. The two bytes are initialized to X'F0F0'.

Phase I also sets appropriate bits in a field which is passed back to the caller of the compiler as the high order byte of register 15. The definitions of these bit settings is defined in the HAL/SDL ICD.

Generation of the internal template is performed during syntax analysis on a token by token basis. As statements are encountered which are required in the template, the tokens from the statements are added to an internal buffer. When a new token will no longer fit in the buffer, the buffer is written and cleared for continuation. Thus, the templates take the form of strings of HAL/S tokens separated by one block. The template statements are continued from one line to the next without regard for statement boundaries, thus producing the template in the most compact form possible.

For the comparison of existing templates with new, generated templates, the generated records are compared character for character with the existing records. Any mismatch is considered to indicate a change in the template.

Templates are never generated using the compressed source format mentioned in Section 2.1. The generated templates conform to the source margins in effect for the compilation (e.g. for an SDL mode compilation, templates are created with source in position 2 through 72 of the records. When template records are written to the OUTPUT6 DD card, the records are padded with blanks or truncated as necessary to conform to the LRECL specification for that DD card.

When a template has been found to have changed, the compiler updates a "Version number" associated with the template. For an existing template, the version number is found on a VERSION compiler directive card at the end of the existing template member. If a new template is needed, the version number is incremented by one and placed on a new VERSION directive card at the end of the generated template. The version number is limited to the range 1 to 255. Upon reaching 255, the next incrementation causes the number to begin again at 1. When no existing template can be located, the version is set to 1.

When templates produced by the compiler are referenced in subsequent compilations by use of an INCLUDE for the template, the version numbers from the referenced templates are emitted into the produced object code on special SYM records which indicate the versions of all external references. In addition, the emitted object code for any compilation unit contains a SYM record indicating the version number of the template created for that compilation unit. This information permits the checking, if desired, of proper integration of separately compiled units by providing information necessary for cross-checking of inter-module references.

2.6 Listing Generation

2.6.1 Primary Formatted Listing

The central printed output of the compiler is the primary source listing. This listing is designed to document the actions taken by the compiler during its generation of an executable form of the user's program. The listing reproduces the user's source program in an indented, annotated format. Additional information, such as block summaries and symbol table listings, are also part of the primary source listing.

The formatting of the primary source listing leads to the documentation of the users program in two ways: 1) variable annotation, and 2) logical indenting.

1) Variable annotation - Each user-defined data symbol, when printed on the primary source listing, receives "marks" appropriate to the type and organization of the symbol. This annotation is that which is defined by the HAL/S Language Specification.

2) Logical indenting - Each statement printed on the primary source listing is formatted and indented to show internal statement structure, and to show the statements' hierarchical and nesting relationships to other statements in the compilation.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

When operating in the SDL, additional information is provided on the primary source listing. The Record Sequence Number, Record Revision Indicator, and Change Authorization fields (see Section 2.1) are printed on the primary source listing next to the statements to which they apply. Additional details of the specific operations performed during SDL operation may be found in the HAL/SDL Interface Control Document.

2.6.2 Error Messages

When compilation errors are detected by Phase I, an error message is printed in the primary listing at the point of detection. All error messages have an identifying code associating with them.

The code is assigned to messages according to a general system which groups errors according to a class and a subclass. Multiple errors within a class/subclass combination are assigned unique numbers within the group. Thus, every possible error in the HAL/S-FC compiler system has a unique identifying code. Appendix C lists the error classification scheme.

The text of all error messages is maintained on a direct access dataset. The compiler retrieves error message text as needed from this dataset. During compilation, the ERROR DD card defines the error message dataset. This file has partitioned organization and contains one member for each error message. The member names are identical to the identifying code assigned to the errors.

The record format of the error library is FB and the logical record length is 80 bytes. The first record of each member defines the severity of that error. The severity is a single EBCDIC number in position one of the first record. The severities and their effects are:

0 = warning (compilation proceeds)

1 = error (further compilation attempted)

2 = severe error (Phase I syntax check proceeds; code generation prevented)

3 = abortive error (compilation halts immediately)

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Within the text of an error message, locations into which specific descriptive information may be placed are denoted by the appearance of two question marks (??). For errors which have this feature, the compiler supplies additional description text (such as the name of an identifier) to make the printed error message as specific and informative as possible.

2.6.3 Block Summaries

The HAL/S-FC compiler provides additional information on the primary listing at the close of HAL/S code blocks. The blocks for which summaries are given are PROGRAM, TASK, FUNCTION and UPDATE.

Information contained in block summaries consists of lists of labels or variable names used in various contexts within the block. The title "BLOCK SUMMARY" begins the list. For all potentially summarized contexts within the block, a descriptive heading is printed followed by the list of names involved. A "*" next to any name in the block summary indicates that the name appears in a context which changes its value. The headings are listed below.

PROGRAMS AND TASKS SCHEDULED
PROGRAMS AND TASKS TERMINATED
PROGRAMS AND TASKS CANCELLED
EVENTS SIGNALLED, SET, OR RESET
EVENT VARIABLES USED
PROGRAM OR TASK EVENTS USED
PRIORITIES UPDATED
EXTERNAL PROCEDURES CALLED
EXTERNAL FUNCTIONS INVOKED
OUTER PROCEDURES CALLED
OUTER FUNCTIONS INVOKED
ERRORS SENT
COMPOOL VARIABLES USED
COMPOOL STRUCTURE TEMPLATES USED
COMPOOL REPLACE DEFINITIONS USED
OUTER VARIABLES USED
OUTER REPLACE DEFINITIONS USED
OUTER STRUCTURE TEMPLATES USED

METRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.6.4 Compilation Layout Summary

Immediately preceeding the Symbol Table printout at the CLOSE of the HAL/S program, there is a compilation layout map, indicating the way in which PROGRAMS, TASKS, PROCEDURES, FUNCTIONS, and UPDATE blocks were defined. The indent level in this printout indicates the nesting level definition of the block shown. This serves to give a quick overview of the compilation structure.

2.6.5 Symbol & Cross Reference Table Listing

The symbol and cross reference table printed at the end of a HAL/S compilation listing provides a detailed accounting of all programmer-defined symbols. The table listing is organized into two parts: a structure template listing and an alphabetized total listing.

Any structure templates defined in the compilation appear first in the symbol and cross reference table. The template names appear in alphabetical order. The body of each template (i.e. the levels defined under the template name) is listed under the template name in the order of definition. This ordering provides a quick reference to the organization of the structure template.

Following any listing of the templates, an alphabetized listing of all programmer-defined symbols is printed. Symbols previously listed as element of a structure template are included in this list. However, the list is completely alphabetized and template organization is not shown. When a particular symbol is independently defined in more than one name scope, the symbol is multiply listed in order of definition.

2.6.6 Built-in Function Cross Reference

Phase I also produces a listing of any HAL/S built-in functions used in a compilation. The printout shows the statement numbers at which the references to the built-in functions occurred.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.6.7 Replace Macro Text

If any HAL/S REPLACE statements were used in the compilation, the text of the macro is printed in the symbol table listing in the attributes and cross reference area.

2.6.8 Unformatted Source Listing

Under control of the "LISTING2" compiler option, Phase I will optionally produce, on the file defined by the LISTING2 DD card, a listing of the input (both SYSIN and INCLUDE) source records as read by the compiler. No special annotation, formatting, or indenting is performed. In the case of input in the SDL compressed format, the LISTING2 option produces the records in their uncompressed format.

2.7 Symbol Table Generation

Phase I is responsible for initial creation of the compiler's internal symbol table. The symbol table consists of a group of arrays which describe all of the properties of declared variables and labels. The capacity of the symbol table is under user control by means of the SYMSIZE compiler option. This table, as created by Phase I, is located in an area common to all compiler phases. Thus, Phase II inherits the initialized table from phase I.

Design of the HAL/S-FC compiler includes, as a basic concept, the use of a Phase I and Phase I/Phase II interface identical to that of the HAL/S-360 compiler. Thus, the description of the internal symbol table to be found in the HAL/S-360 Compiler System Specification, Appendix B.2 is sufficient to define the HAL/S-FC table.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.8 Statement Table Generation

The statement table passes information about executable statements from Phase I of the compiler to Phase III. This information allows Phase III to include statement type and target variable information in the Simulation Data Files.

Due to the use of a common Phase I in the HAL/S-360 and HAL/S-FC compiler systems, the Statement Table description in the HAL/S-360 Specification document is sufficient to describe the HAL/S-FC table. (See Appendix B.3 of that document).

The basic table description includes reference to an "extension" field in which statement memory addresses and/or SRN data is stored. Use of this field is activated by use of certain compiler options:

SRN data is included in the Statement Table if either of the SRN or SDL compiler options are used.

Beginning and ending addresses for individual HAL/S statements are included in the Statement Table when the ADDRS compiler option is used.

The Statement Table is produced on the file specified by the FILE6 DD card. No Statement Table data is communicated via in-memory tables.

2.9 Literal Table Generation

The format of the HAL/S-FC literal table is identical to that used by the HAL/S-360 compiler as described in Appendix B.1 of the HAL/S-360 Compiler System Specification.

The size of the area in which character literal data is stored is under user control via the LITSTRINGS compiler option. This character literal area is communicated to subsequent phases of the compiler through common memory locations.

The portion of the literal table which contains arithmetic literals, bit literals, and pointers to character literals is passed to later phases via the data-set defined by the FILE2 DD card.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.10 HALMAT Creation

HALMAT is the intermediate code medium by which the structure of the compiled HAL/S program is passed to Phase II for code generation. The HAL/S-FC compiler uses the same Phase I as the HAL/S-360 compiler. Therefore, the HALMAT produced by Phase I for either system is the same. A description of HALMAT as used by these compilers can be found in Appendix A of the HAL/S-360 Compiler System Specification.

HALMAT is passed to Phase II through use of auxiliary storage as defined by the FILE1 DD card.

2.11 The Optimizer

The HALMAT produced by Phase I is a direct representation of the HAL/S program being compiled. A separate phase of the compiler exists between Phases I and II which examines and manipulates the HALMAT in order to produce an optimized HALMAT representation. This phase, known as Phase 1.5, is conceptually a part of Phase I. Its operation is transparent to the user as it produces no standard printouts.

The Optimizer performs the following functions:

- Common subexpression elimination, including subscript computations
- Additional literal folding
- Pulling loop invariant subexpressions out of loops
- Replacement of unneeded divisions by multiplications
- Supression of unnecessary matrix transpose operations
- Indicate linear operations on Vectors and Matrices to allow for in-line code
- Indication of procedures which cannot be leaf procedures (as an aid to Phase II)
- Indicate the next use of variables and subexpressions, to assist register allocation in the code generator

These operations are carried out by modifying the HALMAT, literal table, and symbol table.

While the Optimizer is a separate phase, it is conceptually a part of Phase I and is described in the HAL/S-360 Compiler System Specification.

3.0 PHASE II - CODE GENERATION

The code generation phase of the HAL/S-FC compiler has the primary function of producing machine language instructions for the AP-101. Phase II also performs other tasks which are also the subject of this chapter.

This section deals with the following Phase II functions:

- Code Generation
- Naming Conventions
- Printed Data
- Symbol Table Augmentation
- Statement Table Augmentation

3.1 Code Generation

3.1.1 Bases and Conventions

Phase II produces AP-101 machine language instructions which perform the operations indicated by each line of HALMAT received from the syntax and semantic analysis phase. This section describes in detail the ground rules which the code generation phase follows in producing object code. The following terms will be used throughout the ensuing text:

R - A general accumulator (integer or scalar);

X - An indexing register (for subscripting);

B - A base register containing a base address used to compute the effective address of a variable, constant, temporary, or program label.

OFFSET - The constant term which, when subtracted from the actual data address of a variable, yields the address of the 0'th item of the aggregate data collection (note that all HAL subscripts start counting from 1). This is 0 when the variable is a single item.

VAR - The address of a declared non-parameter HAL variable. For addressing purposes, it is actually the base address of the actual data minus the OFFSET. Single valued integer, scalar, or bit input parameters also will use this form.

PAR - The address of a formal parameter passed "by reference". This includes any assigned parameters, plus any input parameters which are not simple integer or scalar variables. Note that PAR actually contains an address.

DELTA - The constant indexing term in a subscript calculation. This term may also reflect the displacement of a structure terminal within a structure template.

OP - Any AP-101 machine instruction.

Note - When VAR or PAR appears in machine instruction constructions, it represents the displacement difference between the data address and the base address contained in the base register B.

3.1.1.1 Register Usage. The following register assignments are used by the code generator:

F0-F5 Used for floating point accumulators and parameters.

F6-F7 Used for floating point accumulators only.

0 Stack register. This register points to the caller's register save area in the run time stack. In addition, all formal parameters, temporaries, and AUTOMATIC variables in REENTRANT procedures are based on this register.

1 Global data addressing register. This register is used to address all of the declared variables and literals within a compilation unit.

2 Work addressing register. This register is used to pass address parameters, dereference NAME variables, and set up any other dynamic addressing.

3 Local addressing register. This register is used in SRS instructions only to address a certain subset of the local data in a block.

4 Linkage register. This register records the return address for all subroutine linkages. It may also be used for an integer accumulator.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

5-7 Used for integer accumulators, index registers, and parameter passage where applicable.

3.1.1.2 Storage Allocation. The HAL/S-FC compiler arranges data in memory such that the least number of base registers need be dedicated in addressing.

Data is grouped into three major categories: single value (constant offset=0), aggregate (character, vector-matrix, structure without copies), and array (including structure with copies). Within in each group, data is ordered such that data requiring the same boundary alignment is adjacent, minimizing the storage lost due to hardware alignment requirements. Within the array group, ordering is further carried on such that multi-dimensional arrays (with larger offsets) come after single dimensional arrays. These above orderings are carried on independently for: 1) program data, and 2) each COMPOOL block contained in the compilation unit. Note that program data includes all variables within the compilation unit including those defined in procedures, functions, or any other block.

Structure templates, unless declared as RIGID, are internally ordered such that the minimum boundary alignment within any node level is required. Template matching requirements guarantee that templates exhibiting identical properties will be identically reordered.

After all groupings are complete, storage assignments are made, with the required base-displacement combinations being generated to properly access the data. Note that the storage addresses assigned refer to the actual data beginning, but the base-displacement address includes the negative OFFSET value. (COMPOOL variables which are not referenced do not have base-displacement addressing generated.)

Note that all formal parameters and all AUTOMATIC variables in a REENTRANT PROCEDURE or FUNCTION are based off the stack register (0).

For arrays, the offset is computed as follows for the number of array dimensions: (N_i is the ith array dimension).

# Dim	Offset
0	0
1	-1
2	$(-1\ N_2)-1$
3	$(((-1\ N_2)-1)N_3)-1$

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

The array OFFSET is then multiplied by the total width of the data type specified. For integers, scalars, bits, and characters, this is the width in halfwords to contain one item of data. For vector and matrix types, this is the width in halfwords for one item times the total number of items in the vector or matrix.

For structures, the OFFSET is 0 if the structure has no copies. If the structure has copies, the offset is -W, where W is the aligned width of one copy of the structure template.

Example:

```
DECLARE  A SCALAR,
         B INTEGER,
         C CHARACTER(7),
         D ARRAY(5) DOUBLE;
DECLARE  E ARRAY(5),
         F ARRAY(3,3) VECTOR,
         G MATRIX;
DECLARE  H DOUBLE,
         I ARRAY(5,5) INTEGER;
```

Alignment	NAME	Location	Base	Displacement	(In Decimal) Offset
Halfword	B	00002	1	0002	0
Fullword	A	00004	1	0004	0
Doubleword	H	00008	1	0008	0
Halfword	C	0000C	1	000C	-6
Halfword	I	00011	1	000B	-2
Fullword	E	0002A	1	0028	-4
Doubleword	D	00034	1	0030	-2
Fullword	G	00048	1	0046	-2
Fullword	F	0005A	1	0040	-26

TRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.1.3 Addressing Concepts. This section describes the general addressing rules for data. To the extent possible, data can be directly addressed via some combination of base register and bit displacement (eleven bits for indexed addressing). This is not possible whenever the data item is a formal parameter other than a simple integer or scalar, or any formal parameter scoped in from an outer to an inner procedure. The skeletal forms given in Section 3.2.2 assume the most commonly used addressing forms. The rules described here should be superimposed upon these skeletal forms to interpret all possible combinations of operations between operands.

Simple Addressing Forms

Simple Variable

```
OP    R, VAR(B)
```

Simple Aggregate Component

(array or vector-matrix)

```
OP    R, VAR+DELTA(X,B)
```

Simple Integer-Scalar formal parameter

```
OP    R, VAR(0)
```

Simple Aggregate formal parameter

```
L     B, PAR(0)
OP    R, DELTA(X,B)
```

NAME Variable in de-reference context

```
LH    B, VAR(B)
OP    R, DELTA(X,B)
```

NAME Variable in de-reference context (ASSIGN formal parameter)

```
L     B, VAR(B)
LH    B, 0(B)
OP    R, DELTA(X,B)
```

REMOTE Variable

```
OP@#  R, ZCON(X,1)
      ZCON  DC  Z(0, VAR, 0)
```


INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Scoped Formal Parameter Addressing Forms

For scoped formal parameters, generation of an effective address requires a loop to trace references back through multiple save areas in the run time stack. In both of the scoped formal parameter sequences below, the offset of 0 in the load instruction at the head of the loop represents the fixed location of the next higher level's register copy. The loop terminates when the nest level of the parameter in question is equal to the nest level of the current save area being referenced.

Scoped Integer-Scalar formal parameter:

```
        LHI     4, <scope number of parameter>
LOOP    LR      2, 0
        L       2, 2(2)
        CH@     4, 9(2)
        BNE     LOOP
        OP      R, VAR(2)
```

Scoped Aggregate or NAME formal parameter:

```
        LHI     4, <scope number of parameter>
LOOP    LR      2, 0
        L       2, 2(2)
        CH@     4, 9(2)
        BNE     LOOP
        LH      2, PAR(2)
        OP      R, DELTA(X,2)
```

Scoped NAME ASSIGN formal parameter:

```
        LHI     4, <scope number of parameter>
LOOP    LR      2, 0
        L       2, 2(2)
        CH@     4, 9(2)
        BNE     LOOP
        LH      2, PAR(2)
        LH      2, 0(2)
        OP      R, DELTA(X,B)
```


INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Address passage addressing forms

For parameter passing to PROCEDURES, FUNCTIONS, and library routines, it is often necessary to pass address pointers instead of data. The following sequences could be used anywhere the instruction LA appears in the generated code sequence (including NAME assignments).

Unsubscripted variable:

```
        LA      R, VAR(B)
```

Subscripted variable:

```
        SLL     X, <index alignment>
        LA      R, VAR(X,B)
```

Unsubscripted REMOTE variable:

```
        L       R, ZCON(1)*
```

Subscripted REMOTE variable:

```
        SLL     X, <index alignment>        or      SLL     R, <index alignment>
        L       R, ZCON(1)*                         A       R, ZCON(1)*
        AR      R, X
```

Unsubscripted variable to REMOTE library:

```
        LA      R, VAR(B)
        IAL     R, X'0400'
```

Subscripted variable to REMOTE library:

```
        SLL     X, <index alignment>
        LA      R, VAR(X,B)
        IAL     R, X'0400'
```

---

* ZCON DC Z(0, VAR, 0).

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Indexing:

The computation for all indexing is done as follows. All constant index terms are factored out from the variable terms. The variable terms are computed according to the natural sequence of unwinding aggregate data. The constant terms are similarly computed to form a DELTA. The variable subscript data being indexed, except for those instructions which perform automatic index alignment. The DELTA is similarly shifted at compile time. If $0 \leq$ DELTA < 2048, it is used in the variable displacement. Otherwise, it is added to X if X is non-zero, or loaded into a newly created X if X is zero (i.e. the subscript contains no variable terms).

3.1.1.4 Condition Codes. The following table lists the allowable relational operations and the resultant condition code - referred to as COND throughout the remainder of this section. Note that the AP-101 conditional branch instructions branch on the "not true" condition.

<OP>	COND
=	3
¬=	4
<	5
>	6
¬ < or > =	2
¬ > or < =	1

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.1.5 ZCONS and the Calling Mechanisms. Throughout the descriptions of generated code of Section 3.1.2, branches to other CSECTs (comsub or library) are generally indicated as:

ACALL <routine name>

The actual implementation of this linkage is to go not directly to the named routine, but instead to branch indirectly through a long address constant (ZCON) located in sector 0 of the machine.

When the target of the branch is a compiler-generated CSECT (a COMSUB), the ZCON referenced will be one created during compilation of the COMSUB. The long indirect address will be in a CSECT named #Znnnnnn (see Section 3.2) which will in turn refer to the real code CSECT.

When the target of the branch is a library routine, the ZCON referenced will be one provided with the library. Its name will be #Qnnnnnn and it will in turn refer to the proper library code CSECT. Certain library routines, for reasons of execution speed, are referenced directly by compiler-emitted code without going through a ZCON. These routines are designated in the BANK0 column of the library documentation. This direct addressing requires that these routines reside either in sector zero or in the same sector as the compiler code which references them.

The use of ACALL in the descriptions implies an external call. In actuality, the instruction generated may be either:

```
SCAL  0, <routine name>
```

or

```
BAL   4, <routine name>
```

depending on whether the library routine has been designated as PROCEDURE or INTRINSIC type.

Some of the parameter setups show the use of P1, P2, and P3 for parameter registers. The following table shows the actual register values for P1, P2, and P3 depending upon the nature of the library routine (see library documentation for specific details).

	P1	P2	P3
Intrinsics	1	2	3
Procedure - P1 used	2	4	7
P1 not used	X	2	4

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

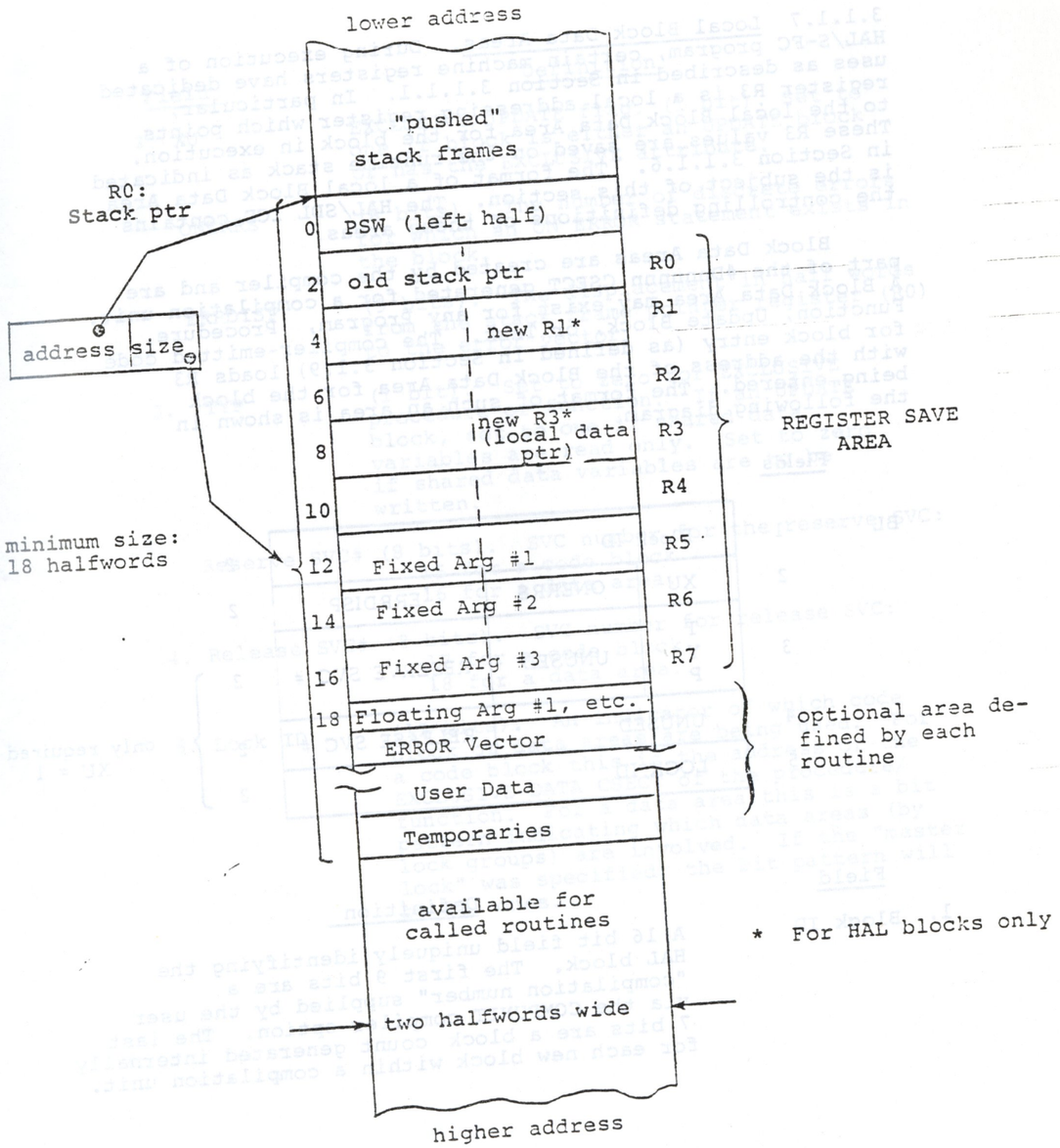
3.1.1.6 The Runtime Stack. The HAL/S-FC compiler system employs a runtime stack mechanism as an integral part of its operation. The stack mechanism is used to provide subroutine linkage areas, temporary work areas, error environments, and to provide reentrancy of code blocks when needed. The actual memory used as a stack space for a given HAL/S process is provided by the flight computer operating system particular stack is made by the flight computer support software linkage editor. The linkage editor determines stack size (and upon special request will create a stack CSECT) from information provided on SYM cards in the modules being link edited. The HAL/S-FC compiler emits the SYM cards as part of its object modules. The runtime library uses a system of macros to generate the properly named DSECT's and SYM entries for stack size computation.

The details of formats and requirements relating to stack generation can be found in the HAL/SDL Interface Control Document. That document also contains the detailed description of the "stack frame", that portion of a total stack which is used by an individual subroutine when that subroutine has been invoked. The description of the basic stack frame is reproduced here for reference.

The active stack frame is pointed to by the pointer in register R0. The back link to the previous stack frame is established when a new level is entered. A pointer, NEW R3, is established for any block with a local data area. If a local data area is not present, e.g. in the case of a HAL/S-FC library routine, NEW R3 is set to zero. See Section 3.1.1.7 for a definition of the local data area.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

STACK LAYOUT

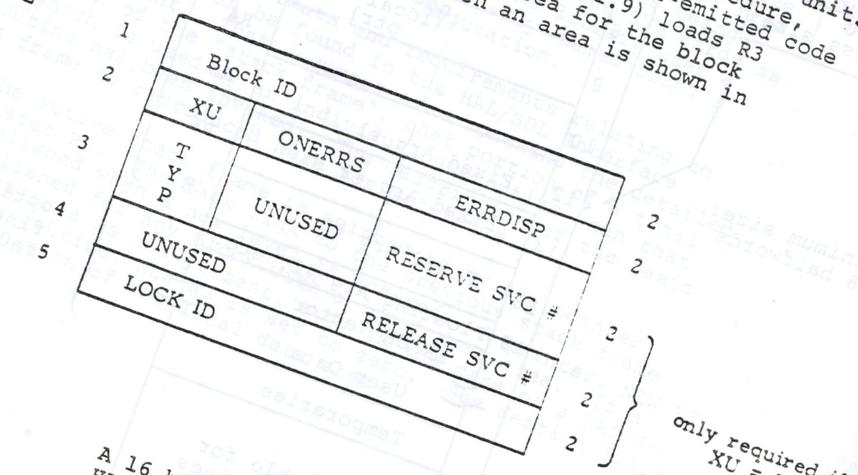


INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.1.7 Local Block Data Areas. During execution of a HAL/S-FC program, certain machine registers have dedicated uses as described in Section 3.1.1.1. In particular, register R3 is a local addressing register which points to the local Block Data Area for the block in execution. These R3 values are saved on the runtime stack as indicated in Section 3.1.1.6. The format of a local Block Data Area is the subject of this section. The HAL/SDL ICD contains the controlling definitions of these areas.

Block Data Areas are created by the compiler and are part of the #Dnnnnnn CSECT generated for a compilation unit. A Block Data Area may exist for any program, procedure, Function, Update Block, or Task. The compiler-emitted code for block entry (as defined in Section 3.1.9) loads R3 with the address of the Block Data Area for the block being entered. The format of such an area is shown in the following diagram.

Fields



Field	Definition
1. Block ID	A 16 bit field uniquely identifying the HAL block. The first 9 bits are a "compilation number" supplied by the user via the COMPUNIT compiler option. The last 7 bits are a block count generated internally for each new block within a compilation unit.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Field	Definition
2. XU	EXCLUSIVE/UPDATE flag. (1 bit). Set to one if block is either an UPDATE block or has the EXCLUSIVE attribute.
ONERRS	(6 bits). The number of discrete errors for which an ON ERROR statement exists in the block.
ERRDISP	(9 bits). The displacement in half words from the stack frame pointer register (R0) to the error vector
3. TYP	(1 bit). Set to zero for EXCLUSIVE procedure or function. If an UPDATE block, set to one if shared data variables are read only. Set to zero if shared data variables are to be written.
Reserve SVC#	(8 bits). SVC number for the reserve SVC: 15 for a code block 16 for a data area.
4. Release SVC#	(8 bits). SVC number for release SVC: 17 for a code block 18 for a data area.
5. Lock ID	(15 bits). An indicator of which code block or data areas are being used. For a code block this is the address of the EXCLUSIVE DATA CSECT of the procedure/ function. For a data area this is a bit pattern indicating which data areas (by lock groups) are involved. If the "master lock" was specified, the bit pattern will be all ones.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.1.8 Parameter Passing Conventions for User-written Routines.

To the extent possible, HAL/S parameters are passed in floating point registers. Scalar parameters are passed in general registers. The following rules describe how the registers are designated, and what they contain for each type of parameter.

General purpose registers 5-7 and floating point registers 0, 2, 4 are available for parameter passing. If the supply of registers is exhausted before the parameter list, the balance of the parameters are passed in memory locations. All parameters are located via the stack register (0).

Allocation of general and floating registers is carried on in parallel. If no scalar parameters exist, no floating point registers will contain parameters.

General purpose registers 5 through 7 are automatically contained in the stack beginning at displacement 1210. Floating point registers are not automatically saved, and it is the responsibility of the called program to do so. Storage locations are reserved in the stack for this purpose as described below. Parameters which cannot be passed in registers are automatically stored in the called procedure's stack by the caller. The allocation of these stack locations is identical to the allocation for floating point values. Note that, unlike ordinary HAL/S variable allocation, parameter allocation is not subject to reordering to minimize alignment conflicts.

The first available stack location is at 18_{10} off the stack register. All parameters are assigned storage in order starting at this point (the exception being parameters contained in general registers 5 through 7, which are allocated space in the register save area as described above). Any necessary alignment is performed as needed.

Arguments are either input type or ASSIGN type. (Input types are those whose values will not be changed by the called routine.) The actual information which is passed for a particular argument is dependent upon the following factors:

- whether the argument is input or ASSIGN;
- whether the HAL/S data type of the argument is an aggregate (i.e. more than one element, as in a matrix);
- whether the argument has any arrayness or structure copies to be passed; and
- whether any arrayness or structure copies are defined via an ARRAY(*) or -STRUCTURE(*) specification.

The following table and list show the information which is passed for an argument with particular attributes.

Argument Type \ Data Type	Integer	Scalar	Bit	Character(*)	Vector	Matrix	Structure
Input (no arrayness or copies)	1	2	3	4	5	6	7
ASSIGN (no arrayness or copies)	8	8	8	4	5	6	7
Input or Assign (with arrayness or copies)	9*	9*	9*	10*	9*	9*	11*

Key — Information Passed

1 — A halfword or fullword of data.

2 — A single or double precision floating point value.

3 — Up to 32 bits of data (halfword or fullword depending upon declared size).

4 — Address of the max-size byte of the character string.

5 — Address of the 0th item in the VECTOR (i.e. 1 item width ahead of the actual vector).

6 — Address of the 0th item as if the MATRIX were a VECTOR of length m x n.

Key	Information Passed
7	Address of the first location in the structure as defined by its template. (Note that item position within a template is subject to compiler reordering unless RIGID is used).
8	Address of the data item.
9	Address of the 0th item of the array.
10	Two items are passed. The first is the address of the 0th array item. The second is the number of halfwords of memory occupied by one character string element (including the halfword containing the max and current size bytes).
11	The address of the first data in the 0th copy.

* If the parameter is declared as ARRAY(*) or -STRUCTURE(*), an additional parameter word is passed containing the value of the unspecified dimension.

For all cases where auxiliary values are allocated for a single parameter (i.e. CHARACTER(*) ARRAY or ARRAY(*)), the parameters (up to 3) must be contiguous. Thus, if more pointers are required than registers are available, then the whole parameter sequence will be pushed into the stack.

Example:

```
P: PROCEDURE(X, Y, I, J, K, Z, C, L);
   DECLARE SCALAR, X, Y, Z DOUBLE;
   DECLARE INTEGER, I, J ARRAY(*), K, L;
   DECLARE CHARACTER(*) ARRAY(*), C;
```


INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Upon entry to this procedure, the stack and registers are as follows:

	— 1 word —		
$R1+12_{10}$	I	unused	also in R5
+14	address of 0th array element of J	unused	also in R6
+16	size of array J	unused	also in R7
+18	X		also in F0
+20	Y		also in F2
+22	K	unused	
+24	1st word of Z		also in F4, F5
+26	2nd word of Z		
+28	address of 0th array element of C	unused	
+30	# HW occupied by one element of C	unused	
+32	Size of array C	unused	
+34	L	unused	

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.2 Integer and Scalar Operations

Nomenclature

The register R is any of the available set of accumulators. The terms I, I_2, S, and S_2 refer to the single and double precision versions of Integer and Scalar values respectively. It is assumed that any implicit precision or type conversions have been accomplished prior to generating the code sequences shown below.

3.1.2.1 **Arithmetic Operators.** Integer and scalar arithmetic operators generally employ two operands, denoted as X and Y. X is assumed to be loaded into register R_x unless otherwise noted. If Y is also in a register, it is represented by the form R_y.

X + Y:

Type	Code	Alternate Code
I	AH R_x, Y	AHI R_x, Y*
I_2	A R_x, Y	AR R_x, R_y
S	AE R_x, Y	AER R_x, R_y
S_2	AED R_x, Y	AEDR R_x, R_y

X - Y:

Similar to X + Y except that the subtract operator is used. (For example, SH in place of AH in the above list.)

(Multiply) X Y:

Type	Code	Alternate Code
I	MH R_x, Y	MIH R_x, Y*
I_2	SLL R_x, 15 M R_x, Y	MR R_x, R_y
S	SRDA R_x, 1 ME R_x, Y	MER R_x, R_y
S_2	MED R_x, Y	MEDR R_x, R_y

Note that the shift operations used in the integer multiplications are required to correctly normalize the result in the proper registers.

Certain constant multipliers are optimized to avoid using actual multiply instructions. They are described below.

* Used if Y is a literal.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	Code	Alternate Code
	I 2^n	SLL R_I, n , n>1 AR R_I, R_I, n=1	
	I_2 2^n	SLL R_I, n , n>1 AR R_I, R_I, n=1	
	X 1	no code for any type	
	S 2	AER R_s, R_s	
	S_2 2	AEDR R_s, R_s	
X/Y:	S	SER R_x+1, R_x+1 DE R_x, Y	SER R_x+1, R_x+1 DER R_x, R_y
	S_2	DED R_x, Y	DEDR R_x, R_y

XY:** The exponentiation is performed by subroutine. The patterns shown for I and S are identical to those which will be generated for I_2 and S_2, except for the obvious differences.

I**I	LH 5, X CVFL 0, 5* LH 6, Y	Argument Setup
S**I	LH 6, Y (see note) LE 0, X	
S**S	LE 2, Y LE 0, X	
	ACALL αPWRβ	Actual Call

where α and β represent the types of operands X and Y respectively:

Type of X	α	Type of Y	β
single precision integer double precision integer	E	single precision integer double precision integer	H I
single precision scalar	D	single precision scalar	E
double precision scalar	D	double precision scalar	D
single precision integer	H*		
double precision integer	I*		

Return is in F0 for α of E or D; in R5 for α of H or I.

* If Y operand is a positive integer literal, the CVFL conversion is eliminated and the PWR routine invoked is αPWRH or αPWRI.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	Code	Alternate Code
X**1:		No code generated.	
X**2:	S	MER R_x, R_x	
	S_2	LER R_T, R_x MER R_x, R_x MER R_T, R_x (result in R_T)	
X**3:	S	MER R_x, R_x LER R_T, R_x MER R_x, R_x MER R_T, T_x (result in R_T)	
X**6:	S	MER R_x, R_x LER R_T, R_x MER R_x, R_x MER R_T, R_x (result in R_T)	

Note: Scalar expressions raised to integer literal powers from 1 to 16 are performed in-line via repeated multiplication, using the binary powers algorithm. The following examples should serve to illustrate the method.

For type S_2, the instruction MEDR is used in place of MER. Two LER's must be used in place of one.

Operation	Type	Code	Alternate Code
+X	I, I_2	No code generated.	
-X	S	LACR R_x, R_x	
	S_2	LECR R_x, R_x LED R_x, X LECR R_x, R_x	

3.1.2.2 Comparison Operators. The full complement of relational operators is allowed for Integer or Scalar operations between single quantities. Only equal or not equal operators are allowed for arrayed comparisons. No logical variables are created by comparisons. Instead, branching to one of two points is used for true/false relations.

Operation	Type	Code	Alternate Code
X < OP > Y:	I	CH R_x, Y BC COND, not-true-label	CR R_x, R_y
	I_2	C R_x, Y BC COND, not-true-label	CER R_x, R_y
	S	CE R_x, R_y BC COND, not-true-label	CEDR R_x, R_y
	S_2	CED R_x, Y BC COND, not-true-label	

Note: For comparisons to the literal 0, the condition code is used directly. If the condition code is not valid, the instruction LR or LER is used to set it.

3.1.2.3 Conversions. Where necessary, conversions are performed in intrinsic or library functions. Some conversions do not require any generation of code.

Integer Conversions

Operation	Type	Code	Alternate Code
*I, I_2 TO S, S_2	I	LH 5, X	argument setup
	I_2	L 5, X	argument setup
		ACALL αTOβ	actual call

TYPE OF INTEGER α	TYPE OF SCALAR β
Single Precision H	Single Precision E
Double Precision I	Double Precision D

* I TO S does not call library routine; instead code generated is:
I CVFL F_x, R_x.

Operation	Type	Code	Alternate Code
I,I_2 TO BIT		No code necessary	
I TO CHAR	I	LH 5, X	
		LA 2, temp-string-area*	
		ACALL HTOC	
	I_2	L 5, X	
		LA 2, temp-string-area*	
		ACALL ITOC	
I TO I_2	I	SRA R_x, 16	
I_2 TO I	I_2	SLL R_x, 16	

Scalar Conversions

Operation	Type	Code	Alternate Code
S,S_2 TO I,I_2	S	LE 0, X	LER 0, R_x
	S_2	LED 0, X	LEDR 0, R_x
		ACALL αTOβ	

TYPE OF SCALAR α	TYPE OF INTEGER β
Single Precision E	Single Precision H
Double Precision D	Double Precision I

Operation	Type	Code	Alternate Code
S,S_2 TO BIT		Same as for scalar to integer	
S TO CHAR	S	LE 0, X	
		LA 2, temp-string-area*	
		ACALL ETOC	

* temp-string-area contains converted string.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	Code	Alternate Code
	S_2	LED 0, X	
		LA 2, temp-string-area*	
		ACALL DTOC	
S TO S_2	S	LE R_x, X	
		SER R_x+1, R_x+1	

3.1.2.4 Assignments. For all assignments, type conversion may take place across the assignment operator. For multiple assignments, the left hand side operands are grouped by data type to minimize the number of conversions performed. The order in which the groups are processed is determined by the following table:

Left Hand Type Ordering	Right Hand Operand Type I	I_2	S	S_2
First	I	I_2	S	S_2
	I_2	Char	Char	Char
	Char	S_2	S_2	S
	S_2	S	I_2	I_2
Last	S	I	I	I

Vector-Matrix Character is always performed before any right hand side conversion is performed.

The following sequences assume that R_x has already had the required integer or scalar conversions performed as described in Section 3.1.2.3.

* temp-string-area contains the resultant string.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type of Y	Code
Y = X;	I*	STH R_x, Y
	I_2	ST R_x, Y
	S	STE R_x, Y
	S_2	STED R_x, Y

R_x is also marked as now containing the value Y. Subsequent usages of Y may use this register in lieu of the copy of Y in memory until such time as the contents of this register are destroyed or a label is generated.

* If X is an integer literal of value 0 or -1, then the following code will be generated:

Y = 0;	I	ZH	Y
Y = -1;	I	SHW	Y

3.1.3 Bit String Operations

3.1.3.1 Bit String Operators. Bit string operators include the following: AND (&), OR (|), and CAT (||). They generally employ two operands, denoted here as X and Y (of lengths N_x and N_y respectively). X is assumed to be loaded into register R_x unless otherwise noted. If Y is also in a register, it is represented as R_y. Note that the & and | operations will pad the bit length of the shorter bit string to the length of the longer bit string.

Operation	Bit Length	Code	Alternate Code
X & Y	N_x, $N_y \leq 16$	NR R_x, R_y	NHI R_x, 'Y'*
	N_x, $N_y > 16$	N R_x, Y	NR R_x, R_y
X | Y	N_x, $N_y \leq 16$	OR R_x, R_y	OHI R_x, 'Y'*
	N_x, $N_y > 16$	O R_x, Y	OR R_x, R_y
X || Y	$N_x \leq 16$	SLL R_x, N_y	
		OR R_x, R_y	
	$N_y > 16$	SLL R_x, N_y	
		O R_x, Y	OR R_x, R_y

* used only when Y is a BIT literal.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.1.3.2 Bit String Comparisons. The only possible relational operators for bit strings, as with bit operators, are = or ¬= (see Section 3.3.1.4). The bit strings are padded to be of equal lengths. No logical variables are created by comparisons. Instead, branching to the "not-true-label" occurs with the "not true" condition.

Operation	Bit Length	Code	Alternate Code
X <OP> Y	N_x, $N_y \leq 16$	CH R_x, Y	CHI R_x, 'Y'*
		BC COND, not-true-label	
	N_x, $N_y > 16$	C R_x, Y	CR R_x, R_y
		BC COND, not-true-label	

3.1.3.3 Component Subscripting. Component subscripting for bit strings consists of shifting and &'ing out unwanted components of the subscripted bit string. The resultant bit string length, N_r, determines a binary mask, whose decimal value is $2^{N_r}-1$, and bit number "I" of the original bit string is the last component of the resultant bit string.

Operation	Bit Length	Code
$X_{subscript}$	N_x	SRL R_x, N_x-I N R_x, mask**
$X_{variable\ subscript}$	N_x	LACR R_I, R_I AHI R_I, N_x SRL R_x, 0(R_I) N R_x, mask**

Examples of Subscript Forms:

Subscript	I	N_r
3 TO 10	10	8
6 AT 11	16	6
9	9	1
8 AT J	J + 7	8
K	K	1

3.1.3.4 Bit Conversions. When necessary, conversions are performed in intrinsic or library functions. Some conversions do not require any generation of code.

* Used only when Y is a Bit literal.
** The mask value is equal to $2^{N_r} - 1$.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

Operation	Bit Length	Code	Alternate Code
BIT TO I		No code necessary	
BIT TO I_2		LH R_x, X	
BIT TO S,S_2	$N_x \leq 16$	SRA R_x, 16	SRA R_x, 16
	$N_x > 16$	LH 5, X CVFL 0, 5	LR 5, R_x
BIT TO CHAR	$N_x \leq 16$	L 5, X ACALL ITOE	LR 5, R_x
	$N_x > 16$	LH 5, X SRL 5, 16 L 5, X	set up of bit-type argument
BIT TO CHAR@<radix>		LA 2, temp-string-area* LHI 6, N_x ACALL BTOC	actual calling sequence

Same as BIT TO CHAR except call to BTOC is replaced as follows:

<radix>	routine
BIN	BTOC
OCT	OTOC
DEC	XTOC
HEX	XTOC

* Temp-string-area contains converted string.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Bit Length	Code	Alternate Code
BIT TO BIT	$N_x > N_y$ $N_y \leq 16$	NHI R_x, 2^{N_y-1}	
	$N_y > 16$	N R_x, mask*	

3.1.3.5 Bit Assignments. The following sequences assume that R_x has already had the required conversions performed as described in Sections 3.1.3.3 or 3.1.3.4.

Operation	Length of Bit String Y	Code
Y = X	$N_y \leq 16$	STH R_x, Y**
	$N_y > 16$	ST R_x, Y**

If the right hand side of the assignment (X) is a BIT literal as described below, and $N_y \leq 16$, then the following code is generated:

Y = BIN'0';	$N_y \leq 16$	ZH Y
Y = BIN(16)'1';	$N_y = 16$	SHW Y

3.1.3.6 Partitioned Bit Assignments. The following sequences assume that R_x has already had the required conversions performed as described in Section 3.1.3.3 or 3.1.3.4. Definitions of I, N_y, and N_r are as described in Section 3.1.3.3.

Operation	Length of Bit String Y	Code
$Y_{subscript}$=X;	$N_y \leq 16$	LH R_x, X LH R_y, Y SLL R_x, N_y-I XR R_x, R_y NHI R_x, mask*** XR R_x, R_y STH R_x, Y

* The value of the mask is 2^{N_y-1}.

** Note: If $N_x > N_y$ and N_y is not exactly 16 or 32, then the following instruction must be added:

N R_x, F'2^{N_r-1}'

*** Mask: The mask used in a bit store is computed as follows:

$(2^{N_r-1})\ (2^{N_x-I})$

In other words, the mask is a sequence of N_y bits shifted left N_x-I bits.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Length of Bit String Y	Code
$Y_{subscript} = X$; (Con't.)	$17 \leq N_Y \leq 32$	L R_Y, Y L R_X, X SLL R_Y, N_Y-I XR R_Y, R_X N R_Y, mask* XR R_Y, R_X ST R_Y, Y

If the right hand side of the assignment (X) is a bit literal containing either BIN'0' or BIN(N_Y)'1', then if $N_Y \leq 16$ and Y is addressable in SRS format, then the following code is generated:

$Y_{11 \text{ TO } 13}$ = BIN'0'; N_Y = 16

$Y_{10 \text{ TO } 12}$ = BIN'111'; N_Y = 16

```
ZB   Y, B'111000'
SB   Y, B'111O000'
```

If $N_Y > 16$ then the following code is generated:

$Y_{13 \text{ TO } 20}$ = BIN'0'; N_Y = 32

$Y_{17 \text{ TO } 20}$ = BIN'111'; N_Y = 32

```
L    Rx', =X'FFF00FFF'
NST  Rx', Y
L    Rx', =X'00007000'
OST  Rx', Y
```

3.1.3.7 Bit Tests.

IF X

IF X_{10}

$N_x = 1$

$N_x = 16$

```
TH   X
BZ   <not true label>
TB   X, B'1000000'
BZ   <not true label>
LH   Rx', X
SRL  R, 6
NHI  R, B'1'
BZ   <not true label>
```

IF ¬X

Same as IF X except BZ changed to BNZ instruction.

* The mask value is computed as: $(2^{N_{r-1}})\ (2^{N_x-I})$.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.4 Character String Operations

3.1.4.1 Character String Operators. The only character string operator is the CAT (||) operating employing two character string operands denoted here as X and Y (of lengths N_X and N_Y respectively). Unless otherwise noted, X is assumed to be loaded into registers R_X. If Y is also in a register, it is represented as R_Y.

Operation	Code
X || Y	LA P3, Y LA P2, X LA P1, temp-string-area ACALL CATV

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.4.2 Character String Comparisons. The full set of relational operators are allowed for character strings (see Section 3.1.1.4 for condition codes). Characters with different lengths are always unequal. No logical variables are created by comparisons. Instead, branching to the "not-true-label" occurs with the "not true" condition.

<OP>	α
=, ¬=	
<, >, <=, >=	C

Operation	Code
X <OP> Y	LA P3, Y LA P2, X ACALL CPRα BC COND, not-true-label

3.1.4.3 Component Subscripting. Component subscripting for character strings consists of setting the initial, N_i, and final, N_f, index values of the subscripted components into registers 5 and 6 respectively, and then branching to the CASP intrinsic.

Operation	Code	Alternate Code
subscripting	LA P2, X LA P1, Y LH 5, N_i LH 6, N_f ACALL CASP	LR 6, 5 { if only 1 component
$Y = X_{subscript}$		

3.1.4.4 Character String Conversions. Where necessary, conversions are performed in intrinsic or library functions.

Operation	Code
CHAR TO I	LA 2, char ACALL CTOH
CHAR TO I_2	LA 2, char ACALL CTOI

Operation	Code
CHAR TO S	LA 2, char ACALL CTOE
CHAR TO S_2	LA 2, char ACALL CTOD
CHAR TO BIT	LA 2, char ACALL CTOB
CHAR TO BIT@<radix>	Same as CHAR TO BIT except call to BTOC is replaced as follows:

<radix>	routine
BIN	CTOB
OCT	CTOO
DEC	CTOK
HEX	CTOX

3.1.4.5 Character String Assignments. Either the receiver variable or the assigned variable in a character string assignment may be subscripted. The possible forms are shown below. When subscripting is used, a partitioning of a character string results. The initial element of this partitioned character string is signified by its index: N_i. Similarly the final element has the index N_f. Some examples of HAL/S subscript forms and the resulting N_i and N_f values are:

Subscript Form	N_i	N_f
1 TO 3	1	3
5 AT 2	2	6

Operation	Code
Y =X	LA P2, X LA P1, Y ACALL CAS*
$Y_{subscript}$=X	LA P2, X LA P1, Y LHI 5, N_{iy} LHI 6, N_{fy} ACALL CPAS*

Operation	Code
Y=X subscript	LA P2, X LA P1, Y LHI 5, Nix LHI 6, Nfx ACALL CASP*
Y subscript=X subscript	LA P2, X LA P1, Y LHI 5, Nix LHI 6, Nfx L 7, H'Niy', Nfy' ACALL CPASP

* For REMOTE data, CASR is called instead of CAS, CASRP for CASP, etc.

3.1.5 Vector Matrix Operations

3.1.5.1 Vector-Matrix Operators. Vector Matrix operators usually operate on two arguments according to the conventions stated in Section 5.2. Since 3-vectors, and 3x3-matrices have special library routines, their code is listed in the column labeled "3-code", while the code for any other vectors or matrices is listed in the "n-code" column.

Operation	Type	n-code	3-code
V1 + V2	single	L R,=H'1,n-1' loop: LE FR,V2(R) AE FR,V1(F) STE FR,temp-area(R) BIX R,loop	L R,=H'1,2' loop:LE FR,V2(R) AE FR,V1(R) STE FR,temp-area(R) BIX R,loop

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	n-code	3-code
V1 + V2	double	L R,=H'1,n-1' loop: LED FR,V2(R) AED FR,V1(R) STED FR,temp-area(R) BIX R,loop	L R,=H'1,2' loop: LED FR,V2(R) AED FR,V1(R) STED FR,temp-area(R) BIX R,loop
V1 - V2	Same as V1 + V2 except that an SE instruction is used instead of an AE instruction for single precision. For double precision, an SED is used instead of an AED.		
-V1	single	L R,=H'1,n-1' loop: LE FR,V1(R) LECR FR,FR STED FR,temp-area(R) BIX R,loop	L R,=H'1,2' loop: LE FR,V1(R) LECR FR,FR STED FR,temp-area(R) BIX R,loop
-V1	double	Same as -V1 single, except that an LED is used in place of the LE instruction.	
V1 ⊘ V2 V1: length n V2: length m results is nxm matrix	single	LA P3,V2 LA P2,V1 LA P1,temp-area LHI 5,n LHI 6,m* ACALL VO6SN	LA P3,V2 LA P2,V1 LA P1,temp-area ACALL VO6S3
V1 ⊘ V2	double	Same as for single precision, except that the routines branched to are VO6DN and VO6D3 for n-vectors and 3-vectors respectively.	
V1 * V2	single	(illegal operation)	LA P3,V2 LA P2,V1 LA P1,temp-area ACALL VX6S3
V1 * V2	double	Same as for single precision, except that VX6D3 is branched to, rather than VX6S3.	

* If both V1 and V2 are the same size, then this instruction will be: LR 6,5.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	n-code	3-code
V1 · V2	single	LA P3, V2 LA P2, V1 LHI 5, n ACALL VV6SN*	LA P3, V2 LA P2, V1 ACALL VV6S3*
	double	Same as for single precision, except that the routines branched to are VV6DN and VV6D3 for n-vectors and 3-vectors respectively.	
V1 · V2 M1 + M2 or M1 - M2 V1 M2 V1: length n M2: nxm	single	Same code as that for adding or subtracting two vectors of length equal to the product of the row size and the column size of M1 and M2. LA P3, M2 LA P2, V1 LA P1, temp-area LHI 5, n LHI 6, m** ACALL VM6SN	LA P3, M2 LA P2, V1 LA P1, temp-area ACALL VM6S3

* The scalar result of the dot product is left in register F0.

** If M2 is of size nxn, then this instruction is: LR 6, 5.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

Operation	Type	n-code	3-code
V1 ∅ M2	double	Same as for single precision, except that the routines branched to are VM6DN and VM6D3 for the general case and the size 3 cases respectively.	
M1 ∅ V1 M1: nxm V1: m	single	LA P3,V1 LA P2,M1 LA P1,temp-area LHI 5,n LHI 6,n ACALL MV6SN	LA P3,V1 LA P2,M1 LA P1,temp-area ACALL MV6S3
M1 ∅ V1	double	Same as for single precision, except that routines branched to are MV6DN (n code) and MV6D3 (3 code).	
V1 ∅ I*, V1 ∅ I2*, V1 ∅ S	single	L R,=H'1,n-1' loop: LE FR,V1(R) ME FR,S STE FR,temp-area(R) BIX R,loop	L R,=H'1,2' loop: LE FR,V1(R) ME FR,S STE FR,temp-area(R) BIX R,loop
V1 ∅ S2	double	L R,=H'1,n-1" loop: LED FR,V1(R) MED FR,S2 STED FR,temp-area(R) BIX R,loop	L R,=H'1,2' loop: LED FR,V1(R) MED FR,S2 STED FR,temp-area(R) BIX R,loop
V1/I... V1/S...		Same as for V1 ∅ I, etc., except than a DE instruction is used instead of ME. (DED is used instead of MED for double precision.)	
I ∅ V1,I2 ∅ V1, S ∅ V1,S2 ∅ V1		Exactly the same as for V1 ∅ I, etc.	
M1 ∅ I,M1 ∅ I2, M1 ∅ S,M1 ∅ S2		Same as for V1 ∅ I, etc., except that the length value (n) is the product of the row size and the column size of M1.	

* Note that in the case of single and double precision integers, they are first converted to scalar form whose value is in F0.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

Operation	Type	n-code	3-code
M1/I,M1/I2, M1/S,M1/S2		Same as for V1/I, etc., except that the length value (n) is the product of the row size and the column size of M1.	
I ∅ M1,I2 ∅ M1, S ∅ M1,S2 ∅ M1		Exactly the same as for V1 ∅ I, etc., except that the length specified in R5 is equal to the product of the row size and the column size of M1.	
M1**i (where i is either a literal or a constant integer)	single	LHI 6, i LA P3, temp-storage-area LA P2, M1 LA P1, temp-storage-area LHI 5, n ACALL MM17SN	Same as for "n-code" where n=3.
M1**i	double	Same as for single precision, except branches to the MM17DN.	
M1**0	single	LA P2, M1 LA P1, temp-storage-area LHI 5, n ACALL MM15SN	Same as for single precision, except branches to MM15DN.
M1**T M1: m x n	double	LA P2, M1 LA P1, temp-stor-age-area LA 5, n LA 6, m ACALL MM11SN	LA P2, M1 LA P1, temp-stor-age-area ACALL MM11S3

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	n-code	3-code
M1**T	double	Same as for single precision, except that the routine branched to is either MM11DN or MM11D3 for n x n matrices and 3 x 3 matrices respectively.	
M1 ∅ M2 M1: k x m M2: m x n	single	LA P3, M2 LA P2, M1 LA P1, temp-area LHI 5, k LHI 6, m* LHI 7, n* ACALL MM6SN	LA P3, M2 LA P2, M1 LA P1, temp-area ACALL MM6S3
M1 ∅ M2	double	Same as for single precision, except that the routines branched to are MM6DN and MM6D3 for the general case and the 3 x 3 case respectively.	

* Either of the instructions may be of the form: LR 6,5 if n=k, etc.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.5.2 Conditional Operators. The only comparison operators allowed for comparing vectors and matrices are = or ~ =. Since these comparisons are done on an element-by-element basis, the same routines that are used for size-n vectors are also used for size n x m matrices which are considered to be vectors of length n x m. No logical variables are created by comparisons. Instead, branching to the "not-true' label" occurs with the "not true" condition.

Operation	Type	n-code	3-code
V1 <OP> V2	single	LA P3, V2 LA P2, V1 LHI 5, n ACALL VV8SN BC COND, not-true-label	LA P3, V2 LA P2, V1 ACALL VV8S3 BC COND, not-true-label
V1 <OP> V2	double	Same as for single precision, except that the routines branched to are VV8DN and VV8D3 for n-vectors and 3-vectors respectively.	
M1 <OP> M2 M1, M2: mxn	single	LA P3, M2 LA P2, M1 LHI 5, mxn ACALL VV8SN BC COND, not-true-label	LA P3, M2 LA P2, M1 LHI 5, 9 ACALL VV8SN BC COND, not-true-label
M1 <OP> M2	double	Same as for single precision, except that the routine branched to is VV8DN.	

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.5.3 Component Subscripting. Possible components of matrices include submatrices, vectors, column vectors, and single components. Possible components of vectors include sub-vectors and single components. The resultant type of component is determined by the subscripts used. Note that double precision operations are not shown - their code is identical except that: a) the called routines will be VV1DN rather than VV1SN, etc; b) the index multiplier is 4 instead of 2. Register 7, when used, contains skip values between elements in partitioned matrices (see Section 3.1.1.3).

Operation*	n-code	3-code
$Y = V_{x_i}$	LE R_x, V_x + 2 * i STE R_x, Y	N.A.
$Y = V_{x_i}$	LH R_I, I LE R_x, Vx(R_I) STE R_x, Y	N.A.
$V_{y_i} = X;$	LH R_I, I LE R_x, X STE R_x, V_y(R_I)	N.A.
Vy_n AT I = Vx_n AT I;	LH R_I, I AR R_I, R_I LA P2, V_x(R_I) LA P1, V_y(R_I) LHI 5, n ACALL VV1SN	LH R_I, I AR R_I, R_I LA P2, V_x(R_I) LA P1, V_y(R_I) ACALL VV1S3

* i indicates integer literal, I indicates integer variable.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation*	n-code	3-code
$M_y = M_{x_m}$ AT I, n AT J assumes M_y is an m by n matrix	LH R$_I$, I MHI R$_I$,<column size of M_x> AH R$_I$, J AR R$_I$, R$_I$ LA P2, M_x(R$_I$) L 7, F'delta,0' LA P1, M_y LHI 5, m LHI 6, n ACALL MM1SNP	<same>
$M_{x*,I} = V_x$;	LH R$_I$, I AR R$_I$, R$_I$ LA P2, V_x LHI 6, 0 LHI 7, delta LA P1, M_x(R$_I$) LHI 5, n ACALL VV1SNP	LH R$_I$, I AR R$_I$, R$_I$ LA P2, V_x LHI 6, 0 LHI 7, delta LA P1, M_x(R$_I$) ACALL VV1S3P

* i indicates integer literal, I indicates integer variable.

3.1.5.4 Conversions. MATRIX/VECTOR conversions are done by considering matrices as vectors, and assigning the required components to the receiver variable. More than 1 argument requires multiple calls to the vector assign routine (as shown in the second sequence below). Use of double precision operands will cause branches to VV1DN. Otherwise, the code is unchanged.

Operation	n-code
VECTOR(M_x) Produces vector of size equal to product of dimension of matrix: n x m.	LA P2, M_x LA P1, temp-area LHI 5, nxm ACALL VV1SN
MATRIX(V_x,V_y,V_z)	LA P2, V_x LA P1, temp-area LHI 5, n_x ACALL VV1SN LA P2, V_y LA P1, temp-area+DELTA1 LHI 5, n_y ACALL VV1SN LA P2, V_z LA P1, temp-area+DELTA2 LHI 5, n_z ACALL VV1SN

* This is an example using several vectors to illustrate the multiple calling of the VV1SN (or VV1S3) routine. It also applies to the VECTOR shaping function.

3.1.5.5 Assignments. Vectors and matrices may be assigned to other vectors and matrices of the same dimensions. In addition, they may have all elements set to zero by a statement of the form:

$\overset{*}{M} = 0$; or $\bar{V} = 0$;

Operation

$V_x = V_y$

Type

single

n-code

```
L     R,=H'1,n-1'
loop: LE    FR,Vy(R)
      STE   FR,Vx(R)
      BIX   R,loop
```

3-code

```
loop: L     R,=H'1,2'
      LE    FR,Vy(R)
      STE   FR,Vx(R)
      BIX   R,loop
```

$V_x = 0$

single

```
L     R,=H'1,n-1'
loop: SER   FR,FR
      STE   FR,Vx(R)
      BIX   R,loop
```

$M_x = M_y$

$M_x = 0$

Same as for $V_x = V_y$, except that the loop count, n-1, is replaced by (m n)-1.

$M_x = 0$

Same as for $V_x = 0$, except that the loop count, n-1, is replaced by (m n)-1.

$V_x = V_y$

$M_x = M_y$

double

Same as single, but use LED, STED sequence instead of LE, STE instructions.

$V_x = 0$

$M_x = 0$

double

Same as single, but use SEDR, STED sequence instead of SER, STE instructions.

For the following operations:

VECTOR/MATRIX ADD
VECTOR/MATRIX SUBTRACT
VECTOR/MATRIX NEGATE
VECTOR/MATRIX-SCALAR PRODUCT
VECTOR/MATRIX-SCALAR DIVIDE
VECTOR/MATRIX ASSIGNMENT

the compiler will attempt to generate in-line code sequences, including as many operations within a single loop as possible. In many cases, the stores into temp-areas's, as shown in the prototype instruction sequences, will not be necessary, unless the resultant VECTOR or MATRIX needs to be passed from one loop to another, or to a library routine. For example:

HAL

```
DECLARE VECTOR,V,W,X,Y,Z;
V = V+(W+X)*Y-Z;
```

Code

```
      L     R,=H'1,2'
L1    LE    FR,W(R)
      AE    FR,X(R)
      STE   FR,templ(R)
      BIX   R,L1
      LA    P3,4
      LA    P2,templ
      LA    P1,temp2
      ACALL VX6S3
      L     R,=H'1,2'
L2    LE    FR,V(R)
      AE    FR,temp2(R)
      SE    FR,Z(R)
      STE   FR,V(R)
      BIX   R,L2
```

THIS PAGE INTENTIONALLY LEFT BLANK

In those cases where in-line code is not generated, the temporary area used to store the result of the last HALMAT operation before an assignment can be eliminated if the vector-matrix statement is of a suitable "form" for optimization and one of four conditions holds. The statement may not have multiple receivers; the single receiver must be a consecutive partition or be nonpartitioned. The precision of the right-hand-side of the statement must match the precision of the receiver. The receiver cannot be a remote variable, and neither the receiver nor the operand(s) of the final HALMAT operation can be name variables, or the terminal of a subscripted structure. Also, variable subscripts on any variables do not allow optimization processing to continue.

Statements that meet these basic requirements can then be checked for the occurrence of a necessary and sufficient condition for optimization. The result of the final operation before the assignment will be stored directly in the receiver if at least one of the following conditions is true:

1) a) The receiver is <u>nonpartitioned</u> and the last operation before the assignment HALMAT is a "Class 3" operation. Class 3 operations include matrix-scalar and vector-scalar multiplication and division, vector-matrix addition and subtraction, vector and matrix negation and the built-in function, UNIT.

b) The last operation is a "Class 1" operation. The class contains only "matrix raised to 0th power". The result, the identity matrix, can be stored directly in any <u>consecutive</u> receiver.

2) The operand(s) are in temporary work areas. Nonconsecutive partitions are moved to work areas when the operands are processed. The result of a previous operation is also in a work area. Operands in work areas are disjoint from the receiver. This is important for "class 2" operations that use the elements of the vector or matrix, vector-vector, and matrix-matrix arithmetic, and matrix transpose and exponentiation (also, the built-in functions, TRANSPOSE and INVERSE). This condition can also hold for class 1 and class 3 operations. If the operation has two operands, <u>both</u> must be in work areas for this condition to be true.

3) The operand(s) are nonidentical to the receiver. A receiver-operand pair is nonidentical if the operand is in a work area, or if neither variable is a formal parameter and the variables have different symbol table references, or if only one of the variables in a formal parameter and the NEST level of the non-parameterized variable is greater than or equal to the NEST level of the parameterized variable (again, symbol table reference cannot be the same).

```
EXAMPLE1: PROGRAM;
   DECLARE MATRIX(3,3), S,T;
   PROC: PROCEDURE(A) ASSIGN(B);
      DECLARE MATRIX(3,3), A,B,C;
      SUBPROC: PROCEDURE(X) ASSIGN(Y);
         DECLARE MATRIX(3,3), X,Y,P,Q;
         Y2 TO 3,* = X2 TO 3,* + C2 TO 3,*;
         B2 TO 3,* = P2 TO 3,* + Q2 TO 3,*;
      CLOSE SUBPROC;
      CALL SUBPROC(A) ASSIGN(C);
   CLOSE PROC;
   CALL PROC(S) ASSIGN(T);
CLOSE EXAMPLE1;
```

where

X&Y are parameters, C is not
NEST_LEVEL(Y)=2,
NEST_LEVEL(C)=1.
Y can be C - cannot assign directly.
P&Q not parameters - ok to assign directly
NEST_LEVEL(P)=2,
NEST_LEVEL(A)=1.

4) The operand(s) are disjoint with the receiver. A receiver-operand pair can be disjoint in two ways. If the pair is nonidentical it is, by default, disjoint. If both they are and the operand are consecutively partitioned, they are disjoint if the partitions do not overlap in any way. If the receiver and the operand have the same symbol table reference (are identical) then the two partitions can be disjoint in either "direction". For example, let A be a 4-by-4 matrix. Then,

$A_{1\ TO\ 2,*} = A_{3\ TO\ 4,*} + \ldots$ and

$A_{3\ TO\ 4,*} = A_{1\ TO\ 2,*} + \ldots$ are both disjoint pairs.

If the receiver and operand are possibly identical, then the pair can only be disjoint if all of the operand partition comes after the receiver partition.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840


```
EXAMPLE2: PROGRAM;
   DECLARE MATRIX(6,3), A,D,E;
   PROC: PROCEDURE(B,C);
      DECLARE MATRIX(4,3), B,C;
      A1 TO 2,* = B3 TO 4,* + C3 TO 4,*;      Pairs A-B & A-C
                                                 disjoint
      A3 TO 4,* = B1 TO 2,* + C3 TO 4,*;      Pair A-B not neces-
                                                 sarily disjoint
   CLOSE PROC;
   CALL PROC(A3 TO 6,*, D3 TO 6,*);           (B1 TO 2,* is really
                                               A3 TO 4,*)
   A3 TO 4,* = D3 TO 4,* + E1 TO 2,*;         A,D,E are, by default,
                                               disjoint because they
                                               are nonidentical
CLOSE EXAMPLE2;
```

If the operation has two operands, both receiver-operand pairs must be disjoint for this condition to be true. The nonidentical and disjoint checks are made at the same time, so this condition also holds if one pair is disjoint by disjoint partitioning and one pair is disjoint by being nonidentical.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.6 Structure Operations

3.1.6.1 Structure Comparisons. Structure comparisons may only be = or ¬=. The comparison is done by comparing corresponding terminal elements of the two structure operands in order of their natural sequence. Each terminal element is referenced by adding the displacement of the element to the address of the structure (see Section 3.1.1.3). No logical variables are created. Instead, branching to the "not-true-label" occurs with the "not-true" condition.

Operation	Code
X <OP> Y	
for each terminal	LA 2, X LA 3, Y LA 2, terminal #1(X) LA 3, terminal #1(Y) LHI 5, width BAL 4, CSTRUC BC COND, not-true-label . . . <same for all terminals> . . . BC 7, true-label

TRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.6.2 Structure Assignment. The assignment of both major and minor structures is done via the BLOCK MOVE algorithm. Generally, this is an MVH code sequence.

Operation	Code
Y = X (neither X nor Y REMOTE)	L R_y,Const1 L R_x,Const2 MVH R_y,R_x . . . DC Y(Y) const1 DC H'n' . . . const2 DC Z(x)
Y = X (X or Y REMOTE)	LA P2,X LA P1,Y LHI 5,width ACALL MSTR

* n is the width of x in halfwords.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.7 Indexing and Arrayed Statements

3.1.7.1 Linear Array Indexing. Linear array indexing is the use of subscripts, on an arrayed data type, to produce a one-dimensional resultant array. In the generated code, only one register - Ra - is needed to keep track of the index value. An initial entry to the array loop (see Section 3.1.7.4), Ra is initialized to a value of 1. On each pass through the loop, Ra is used to define a DELTA value to index the arrayed data (see Section 3.1.1.3). Following this, at the end of the loop the data has been utilized, and is tested to determine if all of Ra is incremented by 1, and is tested to index the arrayed data Ra is any available indexing register. Its contents may not be altered during the course of an arrayed statement. If the index in Ra must be shifted to access the word or doubleword data, it must be moved to another register to perform this shift.

3.1.7.2 Non-Linear Array Indexing. Non-linear array indexing has more than one index which can change values to produce a multi-dimensional resultant array. The actual code generated, though, can only utilize one register - Ra - for indexing. Thus, temporary storage is needed to store all but the inner-most index. As with linear indexing, all index values (both in Ra and temporary storage) are initialized to 1. The DELTA value in memory (see Section 3.1.1.3). Following this, each index defining the index of each arrayed data item is then computed on the basis of the value of Ra and the index values stored value is tested against the size of the corresponding dimension (of the resultant array) to determine if all of the data has been utilized and/or which indices are incremented for the next iteration. An example of this is given in Section 3.1.7.4.

3.1.7.3 Array Indexing. Arrays may be used in their entirety in HAL/S without explicit subscripting (for example assignment of two equally dimensioned arrays). However, the code generated is very similar to that for non-linear indexing, except that the indicies are tested against the size of the corresponding declared dimensions of the arrays, rather than against the size of the corresponding dimensions of the subscripted array. An example of this is shown in the next section.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.7.4 Arrayness and Loop Generation. This section has an example of each possible form of array loops, and how indexing is achieved within them. In general, an array loop consists of the following sections:

a) initialization of index values;

b) computation of address of array element from index value (see Section 3.1.1.3);

c) actual operation to be performed on the array element(s) (i.e. assignment, comparison, etc.);

d) incrementing and testing index values.

It should be noted that non-linear and array indexing produce multiple loops and indices. Since only a single register is available for indexing, temporary storage of index values for outer loops is employed.

Operation	Type	Code	
Linear Indexing:		L 7, =H'1,2'	①
[X] = $[Y]_3$ AT 2	[X]: ARRAY(3) SCALAR [Y]: ARRAY(5) SCALAR DOUBLE	loop:LED 2, Y+4(7) STE 2, X(7)	②
		BIX 7, loop	③

Notes on above example:

① initialize
② assignment
③ increment and test index

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	Code	
Non-Linear Indexing:			
[I] = [V]$_{1,2\ TO\ 3,*:2}$			
	[I]: ARRAY(2,4) INTEGER		
	[V]: ARRAY(2,3,4) VECTOR		
	outer-loop:	L 7, =H'1,1'	1
		ST 7, temp1	
	inner-loop:	L 7, =H'1,3'	2
		LH 6, temp1	
		SLL 6, 2	3
		AR 6, 7	
		MIH 6, =H'3'	
		LH 5, temp1	4
		SLL 5, 2	
		AR 5, 7	
		LE 0, V+100(6)	
		STH 5, temp2	5
		ACALL ETOH	
		LH 6, temp2	
		STH 5, I(6)	
		BIX 7, inner-loop	6
		L 7, temp1	7
		BIX 7, outer-loop	

Notes on the above example:

1. initialization and storage of first index value
2. initialization of second index value
3. indexing of [V]
4. indexing of [I]
5. assignment of scalar value to an integer value
6. incrementing and testing second index value
7. incrementing and testing first index value

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	Code	
[$\overset{*}{M}$] = [$\overset{*}{N}$]	[$\overset{*}{M}$],[N]:	L	R_M, const1
	ARRAY(2,3)	L	R_N, const2
	MATRIX(2,4)	MVH	R_M,R_N
		.	
		.	
		.	
	const1	DC	Y(M)
		DC	H'96'
	const2	DC	Z(N)

Note: 96 is the size of N, (i.e. N is 2x3x2x4x2 = 96 halfwords long).

3.2.3 PROCEDURE/FUNCTION Calls

The PROCEDURE/FUNCTION calling process consists of two parts:

a) argument set up; and

b) the actual branch to the subroutine.

Argument set up uses registers 5-7 as needed for passing integers or bit strings, and/or pointers to vectors, matrices, character strings, arrays or structures. Floating point registers 0, 2, and 4 are similarly used to pass scalar arguments. Once all of these registers are utilized, all remaining arguments are placed in a run time stack for the procedure or function.

The actual code generated sets up the arguments in the order that they appear in the HAL/S PROCEDURE or FUNCTION block definition statement. For example, if the function is:

```
F: FUNCTION(integer_1,scalar_1,scalar_2,vector_1,integer_2);
```

then the registers are loaded in the order:

register 7	using LH or L
register 6	using LA to load the pointer to vector_1
register F2	using LE or LED depending on the precision of scalar_1
register F0	using LE or LED depending on the precision of integer_1
register 5	using LH or L depending on the precision of integer_1

Once all arguments are set up, the actual branch is a BAL or SCAL instruction to the CSECT defined for the procedure or function.

A leaf procedure/function is one which has no stack requirements (i.e. no parameters, no stack temporaries, no local addressable data, no ON ERROR statements, and no intrinsic library calls). Such procedures may be called via BAL R4,<routine name>. These routines are exited using BCR 7, R4.

Operation	Args	Code	Alternate Code
Argument Setup	≤3 non-scalar and ≤3 scalar	LH 7, arg3 LH 6, arg2 LH 5, arg1 LE 4, scalar-arg3 LE 2, scalar-arg2 LE 0, scalar-arg1 ACALL csect-name	L 7, arg3 or LA 7, arg3 L 6, arg2 or LA 6, arg2 L 5, arg1 or LA 5, arg1 LED 4, scalar-arg3 LED 2, scalar-arg2 LED 0, scalar-arg1

Actual Call

Argument Setup	>3 non-scalar and/or >3 scalar	LH R, argn STH R, stack ⋮ LH R, arg4 STH R, stack	① non-scalar stores into stack
		LE FR, scalar-argn STE FR, stack ⋮ LE FR, scalar-arg4 STE FR, stack	② scalar stores into stack
		LH 5, arg1 ⋮ LE 2, scalar-arg2 ACALL csect-name	③

Actual Call

Notes on the above:

①, ② Any additional arguments are generally loaded into any unused register and stored. The actual load op codes may be: L, LH, LA, LE, or LED, depending on the type of argument. Similarly, the stores op codes may be ST, STH, STE, or STED. If the argument already exists in a register, then the code generated will be only a store from that register into the stack.

③ Loading of the first 3 non-scalar, and the first 3 scalar arguments. This is identical to the code shown in the first example above.

3.1.9 Block Definition

3.1.9.1 PROGRAM and TASK Definition.

Operation	Code
PROGRAM or TASK definition	block-name: LA 0, stack-start* LA 1, program-data-csect STH 1, 5(0) IAL 0, stack-size LA 3, local-data-area(1) STH 3, 9(0)

* Omitted if SDL option is turned on.

3.1.9.2 PROCEDURE and FUNCTION Definition.

Both PROCEDURE and FUNCTION definitions are similar to PROGRAM and TASK definitions. However, floating point store instructions are needed to save any scalar arguments passed via registers.

Operation	Code*	Alternate Code
PROCEDURE or FUNCTION definition	block-name	
	For COMSUBS only: LA 1, Program-data-csect STH 1, 5(0) IAL 0, stack-size LA 3, Local-data-area(1) STH 3, 9(0)	
	For EXCLUSIVE only: SVC Local-data-area+2(1)	
	optional: STE 0, stack	STED 0, stack
	STE 2, stack	STED 2, stack
	STE 4, stack	STED 4, stack

* If PROCEDURE or FUNCTION is leaf (has no calls and no stack requirements), then no code is generated for the block entry.

3.1.10 Flow of Control Statements

3.1.10.1 IF...THEN...ELSE.

The code shown below is for the most general form of the IF...THEN...ELSE statement. It is assumed that the condition code from the conditional expression has been generated (see previous subsections on conditional operations).

Operation	Code
IF <cond exp> THEN <...> ELSE <...>	BC cond, else-label then-label: { executable code for THEN clause . . . } BC 7, next-statement else-label: { executable code for ELSE clause . . . } next-statement:
IF <cond exp> THEN <...>	BC cond, next-statement { executable code for THEN clause . . . } next-statement:

3.1.10.2 DO FOR...Loops. The DO FOR loop has two forms: the iterative, and the discrete. They may also cause termination of the loop by use of the clause UNTIL <cond exp>, or WHILE <cond exp>. The use of these clauses is shown for the case of the iterative DO FOR forms where the additional code needed has been labeled "UNTIL code" and "WHILE code". This same additional code is generated for the discrete DO FOR and is placed immediately before the executable code within the DO group (the same process as is illustrated with the iterative DO FOR). Note that the code only shows the use of a single precision integer index; double precision integers, and single or double precision scalars follow the same algorithm with the exception that the corresponding full word, or floating point instructions are used when dealing with the index variable.

Operation		Code	
DO FOR I = a TO b BY c;*		LHI	7, a
	loop-begin:	BC	7, test-label
		⋮	executable code within DO group
	repeat**:	LH	7, I***
		AHI	7, c
	test-label:	STH	7, I
		CHI	7, b
	exit-label:	BC	6, loop-begin
		⋮	code for statement following DO group

* Assumes a, b, and c are literal values.

** This is referenced by the REPEAT statement (see Section 2.3.10.5).

*** This instruction may be omitted if the REPEAT label is not actually used, and the loop index I is already in the designated register.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation		Code		
DO FOR I = a TO b BY c		LHI	7, a	
UNTIL <cond exp>;		BC	7, test-label	
⋮	loop-begin:	TS	temp-area	UNTIL code
END;		BC	4, first-statement*	UNTIL code
		⋮	cond for exp.	UNTIL code
		BC	cond, exit-label	UNTIL code
	first-statement:	⋮	executable code within DO group	
	repeat**:	LH	7, I	
		AHI	7, c	
	test-label:	STH	7, I	
		CHI	7, b	
		BC	6, loop-begin	
	exit-label:	⋮	code for statement following DO group	

* This is done to avoid testing the <cond exp> until after executing through the loop at least once.

** This is referenced by the REPEAT statement (see Section 3.1.10.5).

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation

```
DO FOR I = a TO b by C WHILE <cond exp>
  :
END;
```

Code

```
                LHI   7, a
loop-begin:     BC    7, test-label
                  :  } code for cond exp
                BC    cond, exit-label
                  :  } executable code within DO group
                LH    7, I
test-label:     AHI   7, c
                STH   7, I
                CHI   7, b
                BC    6, loop-begin
exit-label:       :  } code for statement following DO group
```

```
DO FOR I = a1, a2, ..., an
  :
END;
```

```
label-1:  LHI   7, a1
          BAL   4, test-label
label-2:  LHI   7, a2
          BAL   4, test-label
            :
label-n:  LHI   7, an
          LA    4, exit-label
test-label: ST    4, temp-area
          STH   7, I
                  :  } executable code within DO group
repeat*:  L     4, temp-area
          BCR   7, 4
exit-label:       :  } code for statement following DO group
```

* This is referenced by the REPEAT statement (see Section 3.1.10.5).

Operation

```
DO FOR I = I1 TO I2 BY I3
  :
END;
(I1, I2, I3: variables)
```

Code

```
                LH    5, I2
                STH   5, temp-test
                LH    6, I3
                STH   6, temp-incr
                LH    7, I1
                BC    7, test-label
loop-begin:       :  } executable code within DO group
                  :
repeat*:        LH    7, I
                AH    7, temp-incr
test-label:     STH   7, I
                LH    5, temp-incr
                LA    5, loop-begin
                BC    5, positive-test**
                CH    7, temp-test
                BCR   5, 5
                BC    7, exit-label
positive-test:  CH    7, temp-test
                BCR   6, 5
exit-label:       :  } code for statement following DO group
```

* Repeat label (see Section 3.1.10.5)

** This branch is determined by the condition code set by the previous LH 5, temp-incr instruction.

3.1.10.3 DO WHILE/UNTIL. Both of these forms of DO groups are essentially the same except that the DO UNTIL does not test its conditional expression until it has finished executing the code once. In both cases, the condition is tested as detailed in preceeding subsections.

Operation

```
DO WHILE <cond exp>
                                  Code
                        repeat:
                                  .
                                  .  } code for conditional
                                  .      expression
                                BC   cond, exit-label
                                  .
                                  .  } code for statements
                                  .      within DO group
                        exit-label: BC   7, repeat
                                  .
                                  .  } code for statement
                                  .      following DO group
DO UNTIL <cond exp>
                                BC   7, first-statement
                        repeat:
                                  .
                                  .  } code for conditional
                                  .      expression
                        first-statement: BC   cond, exit-label
                                  .
                                  .  } code for statements
                                  .      within DO group
                        exit-label: BC   7, repeat
                                  .
                                  .  } code for statement
                                  .      following DO group
```


3.1.10.4 DO CASE. The DO CASE statement is used to select one of a collection of statements for processing.

Operation

```
DO CASE I;
  <statement 1>
  <statement 2>
      :
  <statement n>
END;
```

Code

```
                LH   Rc, I                    ①
                BC   6, else-case-label
                LA   2, case-vector
                CH   Rc, 0(2)                 ①
                BC   1, else-case-label       ①
                LH   2, 0(Rc, 2)
                BCR  7, 2
else-case-label:
                <else statement code>
                BC   7, exit-case-label
                <statement 1>
                BC   7, exit-case-label
                <statement 2>
                BC   7, exit-case-label
                  :
                <statement n>
exit-case-label:
```

Data

```
                DC   H 'n'
case-vector     DC   Y(statement 1)
                DC   Y(statement 2)
                  :
                DC   Y(statement n)
```

① bounds checks on case number. Omitted if ELSE case not specified.

3.1.10.5 GO TO, REPEAT, EXIT. All of these statements take the form of unconditional branches. It should be noted that REPEAT and EXIT statements may only be used inside DO groups. See Sections 3.1.10.2 and 3.1.10.3 for the locations of the "repeat" and "exit-label" within a DO group.

Operation	Code	
GO TO label	BC 7, label	
REPEAT	BC 7, repeat	"repeat" is the location of the code which determines whether DO group iteration is finished or not.
REPEAT label		
EXIT		
EXIT label	BC 7, exit-label	"exit-label" is the location of the code immediately following the end of the DO group.

3.1.10.6 RETURN. The RETURN statement will branch back from the code for a function to the code immediately following the function's invocation.

Operation	Code	
Procedures & Functions		
RETURN	SVC Local-data-area+3(1)	If EXCLUSIVE only
	SRET 7, 0	normal
Programs & Tasks		
RETURN	BCR 7, 4	leaf PROCEDURE or FUNCTION
	SVC =H'21'	

3.1.10.7 ON ERROR/OFF ERROR/SEND ERROR.

Operation	Code
ON ERROR$_{n:m}$ <stmt>	LA 4, <stmt> STH 4, error table entry 1. LHI 4, <action>* STH 4, error table entry 0 BC 7, next-statement
<stmt>:	<code for stmt>
next-statement:	: } code for next statement
ON ERROR$_{n:m}$ SYSTEM[AND SIGNAL SET RESET <event>]	LA 4, <event> STH 4, error table entry 1 } only if event action phrase present LHI 4, <action> STH 4, error table entry 0
ON ERROR$_{n:m}$ IGNORE[AND SIGNAL SET RESET <event>]	LA 4, <event> STH 4, error table entry 1 } only if event action phrase present LHI 4, <action>* STH 4, error table entry 0
SEND ERROR$_{n:m}$	SVC = X'0014nnmm' ZH error table entry 0
OFF ERROR$_{n:m}$	

* <action> contains action code, error code, and error group as defined in HAL/FCOS ICD.

3.1.11 Built-In Functions

3.1.11.1 Inline Built-in Functions. The following built-in functions emit the inline code shown in the following sequences. In all cases, it is assumed that R_x contains the argument except when a specific load instruction is shown. The results will always be in register R_y.

ABS(arg)

Type	Code

scalar, single	`LE R_y, arg`
	`LECR R_y, R_y`
	`BC 2, *-1`

scalar, double	`LED R_y, arg`
	`LECR R_y, R_y`
	`BC 2, *-1`

integer, single	`LH R_y, arg`
	`LACR R_y, R_y`
	`BC 2, *-1`

integer, double	`L R_y, arg`
	`LACR R_y, R_y`
	`BC 2, *-1`

LENGTH(char)

Type	Code
character string	`LH R_y, char` `NHI R_y, 255`

SIGN(arg)

Type	Code

scalar, single	`LE R_x, arg`
	`LFLI R_y, 1`
	`LER R_x, R_x`
	`BC 5, continue`
	`LECR R_y, R_y`
	`continue: ⋮`

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	Code

	scalar, double	`LED R_x, arg`
		`LED R_y, D'4110000000000000'`
		`LEDR R_x, R_x`
		`BC 5, continue`
		`LECR R_y, R_y`
		`continue: ⋮`

	integer, single	`LH R_x, arg`
		`LFXI R_y, 1`
		`LR R_x, R_x`
		`BC 5, continue`
		`LACR R_y, R_y`
		`continue: ⋮`

	integer, double	`L R_x, arg`
		`L R_y, =F'1'`
		`LR R_x, R_x`
		`BC 5, continue`
		`LACR R_y, R_y`
		`continue: ⋮`

SIGNUM(arg)	scalar, single	`LE R_x, arg`
		`LFLI R_y, 1`
		`LER R_x, R_x`
		`BC 1, continue`
		`BC 4, equal`
		`LECR R_y, R_y`
		`BC 7, continue`
		`equal: SER R_y, R_y`
		`continue: ⋮`

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	Code
SUBBIT$_m$ TO $_n$ (arg) - or- SUBBIT$_{n-m+1}$ AT $_m$ (arg)	integer, single	LH R_x, arg LFXI R_y, 1 LR R_x, R_x BC 1, continue BC 4, equal LACR R_y, R_y BC 7, continue equal: SR R_y, R_y continue: : SRL R_y, 16-n NHI R_y, mask*
	integer, single, or bits of length ≤ 16	
	integer double, or bits of length > 16, or scalar single	L R_x, arg L R_y, =F'1' LR R_x, R_x BC 1, continue BC 4, equal LACR R_y, R_y BC 7, continue equal: SR R_y, R_y continue: : SRL R_y, 32-n N R_y, F'mask'*

* The mask value is: $2^{(n-m+1)}-1$.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Type	Code
SHL (arg, n) SHR (arg, n) XOR(X,Y)	integer	SLL R_{arg}, n SRA R_{arg}, n
	integer Bit, n≤16	LH R_y, Y XR R_x, R_y
	Bit, n>16	X R_x, Y or XR R_x, R_y
MIDVAL(X,Y,Z)	scalar	LE F0, X LE F1, Y MVS F0, Z

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.11.2 Out of Line Functions. Out of line functions require branches to the run time library.

The registers needed for parameter passing, and the name of the library routine branched to, are specified in the tables of Section 5. Examples are given for representative argument types.

Operation	Type of X	Code
COS(X)	scalar, single	LE 0, X ACALL COS
SQRT(X)	scalar, double	LED 0, X ACALL DSQRT
ABVAL(X)	vector(3), double	LA 2, X ACALL VV9D3
TRANSPOSE(X)	matrix(m,n), double	LA P2, X LA P1, temp-area LA 5, n LA 6, m ACALL MM11DN
UNIT(X)	matrix(3,3), single	LA P2, X LA P1, temp-area ACALL MM11S3
RANDOMG	vector(3), single	LA P2, X LA P1, temp-area ACALL VV10S3
TRIM(X)	character	ACALL RANDG LA P2, X LA P1, temp-area ACALL CTRIMV
MAX(X)	array(n)	LA 2, X LHI 5, n ACALL EMAX

3.1.11.3 Shaping Functions. Shaping functions are explicit invocations of type conversion. The generated code for shaping functions has been described in previous subsections where conversions have been described (see Sections 3.1.2.3, 3.1.3.4, 3.1.4.4, and 3.1.5.4).

In addition, when conversion functions are used in a true "shaping" sense, (e.g. MATRIX(<integer array>)), a subroutine is used to move contiguous elements, with possible conversion, to a result location of the desired shape.

Example:

MATRIX(A) where A is a 9 element integer array

```
LA     P2, A1
LA     P1, <result loc>
LHI    6, X'0002' flags*
LHI    5, 9    size
ACALL  QSHAPQ
```

* Flags: 1st 8 bits indicate input data type.
2nd 8 bits indicate output data type.
Values: 0 = H
1 = I
2 = E
3 = D

All REAL TIME statements are implemented by means of a supervisor call (SVC) instruction which has as its address a pointer to a parameter list. The first halfword of this parameter list contains a number which identifies the type of real time call. The remainder of the parameter list varies with the service being requested.

The specific forms of the SVC parameter lists are those described in the HAL/FCOS ICD document.

For real time statements in non-REENTRANT blocks, the SVC parameter lists are in the block's data area. Any invariant portions of the parameter lists are implemented by initialized data. Parts of the parameter lists which are runtime-dependent are created by execution of in-line code preceeding the SVC instruction.

For real time statements in REENTRANT blocks, the SVC parameter lists are dynamically created in the stack by executable code preceeding the SVC instruction.

3.1.12.1 WAIT Statement. The WAIT statement may use registers 0, 1 to contain a double precision time value specified in seconds. If the UNTIL option is specified, the time value is expressed as mission elapsed time. Any other times are 'delta-time' from the current mission elapsed time. If a time value is not specified in the WAIT statement, then the registers will not be affected.

Operation	Type	Code
WAIT n	n: literal	LED 0, D'floating point form of n' SVC parameter-list
WAIT X	X: scalar double	LED 0, X SVC parameter-list
WAIT FOR DEPENDENT		SVC parameter-list
WAIT FOR X	X: event value	SVC parameter-list
WAIT UNTIL X	X: scalar double	LED 0, X SVC parameter-list

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.12.2 CANCEL, TERMINATE Statements.

Operation	Code
CANCEL CANCEL<task id> TERMINATE TERMINATE<task id>	SVC parameter-list

3.1.12.3 SIGNAL, SET, RESET Statements.

Operation	Type	Code
SIGNAL<event var>	latched or unlatched	SVC parameter-list
SET<event var>	latched	
RESET<event var>	latched	

3.1.12.4 UPDATE PRIORITY Statement.

Operation	Type	Code
UPDATE PRIORITY TO i UPDATE PRIORITY<taskid> TO i	i: integer	SVC parameter-list

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.12.5 SCHEDULE Statement. In the following code generation sequences, a schematic representation of possible SCHEDULE statement forms has been used. The symbol [ ] means that one of the contained elements may appear in the statement form without affecting the generated code. The symbol { } means that one of the contained elements must be included in the statement form - but which one does not affect the code generated.

In general, the code differs only when time values are specified in the SCHEDULE statement. This requires that the time values be specified in double precision format in certain registers as shown below.

Operation	Code
SCHEDULE<label>[ON<event exp>]PRIORITY(I)[DEPENDENT][WHILE<event exp>][UNTIL<event exp>]	SVC parameter-list
SCHEDULE<label>{AT X / ON X}PRIORITY(I)[DEPENDENT][WHILE<event exp>][UNTIL<event exp>]	LED 0, D'X' SVC parameter-list
SCHEDULE<label>[ON<event exp>]PRIORITY(I)[DEPENDENT],REPEAT{AFTER X / EVERY X}	LED 2, D'X' SVC parameter-list
SCHEDULE<label>[ON<event exp>]PRIORITY(I)[DEPENDENT]UNTIL X	LED 4, D'X' SVC parameter-list
SCHEDULE<label>{AT X / ON X}PRIORITY(I)[DEPENDENT],REPEAT{AFTER Y / EVERY Y}[WHILE<event exp>][UNTIL<event exp>]	LED 0, D'X' LED 2, D'Y' SVC parameter-list

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Code
SCHEDULE<label>{AT X / ON X}PRIORITY(I)[DEPENDENT]UNTIL Y	LED 0, D'X' LED 4, D'Y' SVC parameter-list
SCHEDULE<label>[ON<event exp>]PRIORITY(I)[DEPENDENT],REPEAT{AFTER X / EVERY X}UNTIL Y	LED 2, D'X' LED 4, D'Y' SVC parameter-list
SCHEDULE<label>{AT X / ON X}PRIORITY(I)[DEPENDENT], REPEAT{AFTER Y / EVERY U}UNTIL Z	LED 0, D'X' LED 2, D'Y' LED 4, D'Z' SVC parameter-list

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.1.13.1 Initiation. Initiation of either READ, READALL, or WRITE statements consists of a branch to the IOINIT library routine. Register 1 contains the I/O channel number, and register 0 indicates the type of I/O to be initiated.

Operation	Type	Code
READ(n)...		LHI 6, n LHI 5, 0 ACALL IOINIT
READALL(n)...		LHI 6, n LHI 5, 1 ACALL IOINIT
WRITE(n)...		LHI 6, n LHI 5, 3 ACALL IOINIT

3.1.13.2 Input. In all cases, the code sequences below follow the I/O initiation process described in the previous subsection. It is assumed that any conversions have been done previous to the code sequences shown; the resultant type determines which type of code sequence is generated. Note that vector and matrix partitioning require that the first element of the partition be known; additionally, matrices require a DELTA value to be known to skip over those elements (in the "natural sequence") which are not part of the resulting partitioned matrix (see Section 2.1.1.3).

Operation	Type	Code
READ( )..., I, ...	integer, single	. . . initiation LA 2, I ACALL HIN
	integer, double	. . . initaation LA 2, I ACALL IIN
READ( ),..., S, lll	scalar, single	. . . initiation LA 2, S ACALL EIN
	scalar, double	. . . initiation LA 2, S ACALL DIN
READ( )..., V, ...	vector(n);single	. . . initiation LA 2, V XR 7, 7 LHI 5, 1 LHI 6, n ACALL MMRSNP

Operation	Type	Code
READ( )..., V, ...	partitioned vector of length n whose first element is located at 'V+displacement'	. } initiation LA 2, V+displacement XR 7, 7 LHI 5, 1 LHI 6, n ACALL MMRSNP
READ( )..., M, ...	vector(n); double (partitioned or not partitioned)	same except branches to MMRDNP
READ( )..., M, ...	matrix(m,n); single partitioned matrix whose resultant size is mxn, first element is M+displacement.	. } initiation LA 2, M XR 7, 7 LHI 5, m LHI 6, n ACALL MMRSNP
	matrix(m,n); double (partitioned or not partitioned)	. } initiation LA 2, M+displacement LHI 7, DELTA LHI 5, m LHI 6, n ACALL MMRSNP
READ( )...,C,... or READALL( )...,C,...	character string	Same except branches to MMRDNP
READ( )..., C_m TO n'... or READALL( )..., C_m TO n'...	partitioned character string	. } initiation LA 2, C ACALL CIN
		. } initiation LA 2, C LHI 5, m LHI 6, n ACALL CINP

Operation	Type	Code
READ( )..., C_n,... or READALL( )..., C_n,...	single partitioned character string	. } initiation LA 2,C LHI 5,n LR 6,5 ACALL CINP
READ( )..., B,...	bit string (of length n)	. } initiation LHI 6,n ACALL BIN* BC 4,around ST 6,B around: .

Arrayed Input — The actual code generated depends on the type of array. Thus, the code will consist of an array loop (see Section 2.1.7.3) which contains the proper code for inputting of each array element using the code shown above (corresponding to the array element type).

3.1.13.3 Output. In all cases, the code sequences below follow the I/O initiation processes described in Section 2.1.12.1. It is assumed that any conversions have been done previous to the code sequences shown; the resultant type determines which type of code sequence is generated. Note that vector and matrix partitioning require that the first element of the partition be known; additionally, matrices require a "delta" value be known to skip over those elements (in the "natural sequence") which are not part of the resulting partitioned matrix.

Operation	Type	Code
WRITE( )..., I,...	integer, single	. } initiation LH 5, I ACALL HOUT
	integer, double	. } initiation L 5, I ACALL IOUT

* BIN returns the bit string input in register R6.

Operation	Type	Code
WRITE( )..., S,...	scalar, single	. . . } initiation LE 0, S ACALL EOUT
WRITE( )..., V,...	scalar, double	. . . } initiation LED 0, S ACALL DOUT
WRITE( )..., V, ...	vector(n); single partitioned vector of length n whose first element is located at 'V+displacement'	. . . } initiation LA 2, V XR 7, 7 LHI 5, 1 LHI 6, n ACALL MMWSNP
WRITE( )..., M,...	vector(n); double (partitioned or non-partitioned) matrix(m,n); single	. . . } initiation LA 2, V+displacement XR 7, 7 LHI 5, 1 LHI 6, n ACALL MMWSNP Same except branches to MMWDNP
WRITE( )..., M,...	partitioned matrix of resultant size mxn whose first element is M+displacement	. . . } initiation LA 2, M XR 7, 7 LHI 5, m LHI 6, n ACALL MMWSNP
		. . . } initiation LA 2, M+displacement LHI 7, delta LHI 5, m LHI 6, n ACALL MMWSNP

Operation	Type	Code
	matrix(m,n); double (partitioned or not partitioned)	same except branches to MMWDNP
WRITE( )..., C,...	character string	. . . } initiation LA 2, C ACALL COUT
WRITE( )..., C_m TO n	partitioned character string	. . . } initiation LA 2, C LHI 5, m LHI 6, n ACALL COUTP
WRITE( )..., C_n,...	single partitioned character string	. . . } initiation LA 2, C LHI 5, m LR 6, 5 ACALL COUTP
WRITE( )..., B,...	bit string (of length n)	. . . } initiation L 5, B LHI 6, n ACALL BOUT

Arrayed Output

The actual code generated depends on the type of array. Thus, the code will consist of an array loop (see Section 2.1.7.3) to cause iterative outputting of each array element using the code shown above (corresponding to the array element type).

3.1.14 NAME Operations

3.1.14.1 NAME Comparisons.

NAME comparisons may only be = or ¬=.

Operation	Code
NAME(X) <OP> NAME(Y) X,Y - NAME Variables	LH R_x, X LH R_y, Y CR R_x, R_y BC COND, not-true-label BC 7, true-label
NAME(X) <OP> NAME(Y) X - declared variable Y - NAME variable	LA R_x, X LH R_y, Y CR R_x, R_y BC COND, not-true-label BC 7, true-label

3.1.14.2 NAME Assignments.

The variable Y in the following examples may only be a NAME variable. The variable X may be either an actual or NAME variable having declared properties identical to Y.

NAME(Y) = NAME(X); where X is declared variable	LA R_x, X STH R_x, Y
NAME(Y) = NAME(X); where X is NAME variable	LH R_x, X STH R_x, Y

3.1.15 %MACROS

The following %MACROS are recognized by the HAL/S-FC compiler and produce the indicated code.

3.1.15.1 %SVC.

Operation	Code
%SVC(α)	SVC α

3.1.15.2 % NAMECOPY.

This operation works in the same manner as NAME assignments except that the operands must be structures, but not necessarily having identical properties.

Operation	Code
%NAMECOPY(Y,X); where X is actual variable	LA R_x, X STH R_x, Y

3.1.15.3 %COPY.

Operation	Code
%COPY(X,Y)	L R_x, =Y(X, size of Y) L R_y, =Z(Y) MVH R_x, R_y
%COPY(X,Y,n)	L R_x, =Y(X,n) L R_y, =Z(Y) MVH R_x, R_y
%COPY(X,Y,5);	L R_y, Y ST R_y, X L R_y, Y+2 ST R_y, X+2 LH R_y, Y+4 STH R_y, X+4

3.1.16 NONHAL References

Definition and use of the NONHAL construct in the HAL/S-FC compiler system results in an unimplemented feature message from the code generator.

3.1.17 Block Move Algorithm

For assignments involving arrays, structures, or vectors and matrices, the block move algorithm will be applied when:

1) both source and destination data specifications occupy contiguous storage;
2) the data types for source and destination match in type and precision; and
3) neither source nor destination variables are NAME variables.

The default block move sequence is as follows:

```
        L     Rx,=Z(X)
        L     Ry,=Y(Y,size of X)
        MVH   Ry,Rx
```

In certain block move applications where the following is true:

a) variable indexing is not specified;
b) no NAME or ASSIGN parameters are specified;
c) source and data word alignments match and 8 or fewer halfwords are being moved, or two or fewer halfwords are being moved;

then the MVH sequence is replaced by the appropriate number of LED/STED (4 halfwords apiece), L/ST (2 halfwords apiece), and LH/STH instruction pairs to accomplish the movement.

The following examples illustrate:

```
DECLARE ARRAY(3),
    A,B,C INTEGER,D INTEGER;
DECLARE ARRAY(3) INTEGER,
    E,F;                    /*E,F on odd boundary*/

  A=B;
      LED     0,B
      STED    0,A
      LE      0,B+4
      STE     0,A+4
  C=D;
      L       S,D
      ST      S,C
      LH      S,D+2
      STH     S,C+2
  E=F;
      LH      5,F
      STH     5,E
      L       5,F+1
      ST      5,E+1
B=E;
```

uses MVH since >2 halfwords and alignments not matching.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.2 Object Code Naming Conventions

Each successful HAL/S compilation produces several named control sections (CSECTs). The CSECT names are derived according to the following rules:

a) HAL/S compilation unit names are transferred to the emitted object code by using only the first six characters of the HAL/S name. The name will be padded or truncated to six characters where necessary.

b) Any occurrence of the underscore character (_) in the first six characters of a PROGRAM, PROCEDURE, FUNCTION, TASK, or COMPOOL is eliminated. The resulting characters are joined together to produce the characteristic name of the compilation unit (e.g. A_B_C becomes ABC). Additional characters are placed on the front of the resultant name to form the final name for each of the individual situations in which the name is used. All CSECT names therefore take the form:

```
ccNNNNNN
```

where the value of cc for individual cases is:

```
PROGRAM                          :  $0
TASKs                            :  $c c=(1-9, A-Z)
COMSUBs                          :  #C
Internal procs                   :  an a=(A-Z), n=(0-9)
DECLAREd data                    :  #D
COMPOOL                          :  #P
Process Directory Entries:       #E
Z-con to comsub                  :  #Z
Remote data                      :  #R
Exclusive data                   :  #X
```

In addition to CSECT's produced by the compiler, the HAL/S-FC system defines other CSECT's, some of which are referenced by compiler-emitted code. These CSECT types and their naming conventions are:

```
Z-con to library routine:   #Q
Data for library routines:  #L
```


INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.3 Printed Data From Phase II

Under control of the LIST compiler option, Phase II will produce a formatted, mnemonic listing of the object code produced for the compilation unit. In addition to the assembler-type mnemonic instruction listing, a full hexadecimal listing of the emitted code is also produced.

This object code listing is normally appended to the Phase I primary source listing as defined by the SYSPRINT DD card. However, use of the SDL compiler option in addition to the LIST option causes the object code listing to be produced through the OUTPUT7 DD card. The listing thus produced is compatible with the ABSLIST function of the AP-101 Link Editor. The HAL/SDL ICD contains the detailed description of the ABSLIST format.

3.4 Symbol Table Augmentation

Phase II inherits an initialized symbol table from Phase I. In the course of generating code, Phase II makes additions to the symbol table which are inherited, in turn, by Phase III. These additions are generally in the area of data addressing.

Information is added in two of the symbol tables parallel arrays:

- The SYT_ADDR array is filled with data offset information indicating the relative location of data items within CSECTs.
- The EXTENT array is filled with information about the size of the storage allocated to individual data items.

3.5 Statement Table Augmentation

Phase III inherits, in a secondary storage device, the statement table produced by Phase I. If the ADDRS compiler option is in effect, Phase I leaves room in the statement table for beginning and ending addresses of individual HAL/S statements. This information is filled in by Phase II after the generation of the executable code has been performed. The completed statement table is then left for use by Phase III.

4.0 PHASE III - SIMULATION DATA FILE GENERATION

Phase III of the HAL/S-FC compiler has the primary function of providing Simulation Data Files (SDFs) for each unit of compilation. Phase III also produces user-oriented printouts upon special request. This section deals with the following Phase III functions:

- SDF generation
- Printed data

4.1 SDF Generation

Phase III synthesizes the SDF for a compilation unit from data received from previous Phases of the compiler. This data is primarily in two areas: a) The symbol table, created by Phase I and augmented by Phase II, and b) The statement table similarly created by Phase I and II.

The detailed format of an SDF is controlled by the HAL/SDL Interface Control Document. The reader is referred there for details of SDF design beyond the overview presented in the next section.

4.1.1 Overall SDF Design

A Simulation Data File (SDF) is an organized and directoried collection of block, symbol, and statement data which is created by the HAL compiler from a single unit of compilation and stored in a permanent form for later use by simulation processors.

There are basically three types of information contained in an SDF. These are:

1) Symbol Data - contains the attributes of HAL symbols (labels and variables) such as name, class and type, relative core address, number of bytes in core occupied, etc. Also contains arrayness and dimensionality for arrayed variables, template linkages for elements of structures, and cross-reference information listing all statements within the compilation unit that may assign values to the symbol.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2) Statement Data - contains the attributes of HAL statements such as type, Statement Reference Numbers (SRNs) if specified by the user, indices for all labels attached to each statement, and indicies for all variables which may be assigned values by that statement. Also may optionally contain the relative core addresses of the first and last executable instructions emitted for that statement.

3) Block and Directory Data - contains information about each HAL block and the symbols and statements contained within that block, plus information concerning the layout and organization of the SDF which minimizes the time needed to access desired data entries.

An SDF is produced for all compilation units unless suppressed by the user (the TABLES/NOTABLES option). In the case of COMPOOL compilations, the SDF becomes somewhat simplified, having no executable statements and, consequently, no cross-reference data for its symbols.

SDFs are created as members of Partitioned Data Sets (PDSs) and are assigned names of the form ##CCCCCC, where CCCCCC is the first six characters of the compilation unit name with any and all underscore characters removed. (Example; the SDFs for the compilation units SAMPLER and TEST_SAMPLE would be assigned the names ##SAMPLE and ##TESTSA, respectively). The members are written in fixed record format with a block size and logical record length of 1680.

The structure of the SDF will support three efficient types of access:

1) Given the name of a symbol, and the name of the block in which it was declared, obtain the attributes of the symbol.

2) Given a Statement Reference Number (SRN), obtain the attributes of the statement.

3) Given an Internal Statement Number (ISN), obtain the attributes of the statement.

In access methods 1) and 2), the SDF directory plays a key role. When the symbol name and its block are given, the directory will identify which particular physical record of the SDF contains the corresponding fixed-length Symbol Node. Once this record has been read into core, a simple and fast binary search will locate the symbol node which in turn "points" directly to the attributes of the symbol which are contained within a variable-length Symbol Data Cell. A virtually identical procedure can be used to locate statement data when the SRN is given. In this case, the fixed-length nodes involed in the binary search are called Statement Nodes, and their corresponding variable-length data cells are called Statement Data Cells.

In contrast to access methods 1) and 2), which require directory help followed by binary searches, method 3) is direct. This is because there is a one-to-one correspondence between the ISN (compiler-generated Internal Statement Number) and the order of the Statement Nodes. The HAL/SDL ICD contains detailed descriptions of the SDF organization.

4.2 Phase III Printed Data

For each invocation of Phase III, a set of tabular data is printed. The information presented deals with parameters relating to the SDF produced, such as number of SDF pages, numbers of block, symbol, and statement nodes, etc.

In addition to the information which is always printed, two optional printouts are available. Under control of the TABLST compiler option, the user may request that symbolic, structured dump of the SDF be provided. In addition, under control of the TABDMP compiler option, the user may request that the contents of the SDF be displayed in a hexadecimal format, page by page.

5.0 RUN TIME LIBRARY

5.1 Introduction

This section describes the HAL/S-FC runtime library as used to support the HAL/S-FC compiler. The material is organized to present both general design concepts and detailed interface and algorithm information. Following an introductory discussion of general conventions used throughout the library, descriptions of the individual routines are grouped according to the basic type of the routine. Each group is introduced by a quick-reference chart containing basic interface data.

5.2 Basics and Conventions

5.2.1 Origin and Format

The HAL/S-FC compiler comes supplied with a run time library. The library is a partitioned dataset (PDS) in IBM AP-101 load module format. Each primary member of the library was generated by assembling the identically named member of a source library consisting of statements written in AP-101 Basic Assembler Language (BAL). Some source library members produce more than one entry point, in which case load module library ALIAS names are generated for each entry. A macro library was used to standardize frequently used sequences of source code.

5.2.2 Purpose

The run time library is used to supply routines, data and interfaces which are needed to execute a HAL/S program or group of programs, which are not produced by the compiler's code generator. Most of the library consists of subroutines which are called from compiler generated code in a HAL statement.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

5.2.3 Intrinsics and Procedure Routines

The library routines are divided into two groups: intrinsics and procedures. The main distinction is that procedure routines save the passed contents of all fixed point registers, while intrinsics do not. For this reason, a procedure can call another routine (e.g. vector (VV10S3) a procedure calls SQRT), but an intrinsic cannot. Intrinsics magnitude do not have a new stack level and therefore do not have any **stack work areas. Because intrinsics do not save all passed** contents of fixed point registers, they cannot restore them, and must not destroy any register contents that must be returned to the calling program. Expansions of the macros within intrinsics routines are different from the expansions within procedure routines.

5.2.4 Register Conventions in Run Time Library Routines

5.2.4.1 General Purpose Registers R0-R7.

R1-R3, R5-R7:

free use;

R4 : return address during calling and exiting intrinsics, otherwise free use;

R0 : stack base;

Parameters : Intrinsics: any or all of R1, R2, R3, R5, R6, R7 can be used for parameter passing.
procedures: any or all of R2, R4, R5, R6, R7 can be used for parameter passing.

5.2.4.2 Floating Registers F0-F6.

F0-F4 : free use;

F6 (F7) : may be used only if saved and restored at entry and exit;

Parameters : depending on the individual routine, any or all of F0-F4 can be used for parameter passing.

Only F6 is guaranteed constant across procedure calls.

5.2.4.3 Interface Conventions.

In addition to the parameter passing conventions summarized in general form in the previous two sections and given in detail in the individual library routine descriptions, the compiler has information defining the linkage conventions and register usage for each routine. This section contains that information in a list formatted in four columns as follows:

NAME — The primary or secondary entry point name.

CALL TYPE — Either PROCEDURE or INTRINSIC to distinguish between routines which must be called via the SCAL instruction and those that must be called using BAL.

BANK0 — YES indicates that the routine will always reside in Sector 0 of the GPC and may therefore always be called directly (no ZCON needed). NO indicates that the routine may reside in a sector other than 0 and must therefore be called via a long indirect address constant (ZCON).

Registers assumed to be modified — A list of registers which the compiler assumes to be modified across a call to the routine. Any registers not listed may be assumed to remain unmodified and therefore to maintain their previous contents.

Any modifications to compiler or library should be made carefully so as to maintain this interface properly.

NAME	CALL TYPE	BANKO	REGISTERS ASSUMED TO BE MODIFIED
ACCS	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
ASIN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
ACCSH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
ASINH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
ATANH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
BTOC	PROCEDURE	NC	R1,R2,R3,R4,R5,R6,R7
CASPV	INTRINSIC	NC	R1,R2,R3,R4,R5,R6
CASP	INTRINSIC	NC	NONE
CASRPV	INTRINSIC	NC	NONE
CASRP	PROCEDURE	NC	NONE
CASRV	PROCEDURE	NC	R1,R2,R3,R4,R5
CASR	PROCEDURE	NC	R1,R2,R3,R4,R5
CASV	PROCEDURE	NC	R1,R2,R3,R4,R5
CAS	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1
CATV	INTRINSIC	NC	NONE
CAT	INTRINSIC	NC	R5,F0,F1,F2,F3,F4,F5
CIN	INTRINSIC	NC	F0,F1
CINDEX	PROCEDURE	NC	F0,F1
CINP	PROCEDURE	NC	NONE
CLJSTV	PROCEDURE	NC	NONE
CCUTF	PROCEDURE	NC	F0,F1
COUT	PROCEDURE	NC	F0,F1
CPAS	PROCEDURE	NC	F0,F1
CPASP	PROCEDURE	NC	F0,F1
CPASR	PROCEDURE	NC	R2,R3,R4,R5,R6
CPASRP	PROCEDURE	NC	R2,R3,R4,R5,R6
CPB	PROCEDURE	NC	NONE
CPBC	INTRINSIC	NC	F0,F1
CPBA	INTRINSIC	NC	F0,F1,F2,F3,F4,F5
CRJSTV	PROCEDURE	NC	R5,F0,F1
CSHAPQ	PROCEDURE	NC	R5,F0,F1
CSLD	PROCEDURE	NC	R5,F0,F1
CSLDP	PROCEDURE	NC	R5,F0,F1
CPSLD	PROCEDURE	NC	R5,F0,F1
CPSLDP	PROCEDURE	NC	R5,F0,F1
CSST	PROCEDURE	NC	R5,F0,F1
CSSTF	PROCEDURE	NC	NONE
CPSST	PROCEDURE	NC	R2,R3,R4,R5,R6
CPSSTF	PROCEDURE	NC	R5,F0,F1
CSTR	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
CSTRUC	INTRINSIC	NC	F0,F1,F2,F3,F4,F5
CTCB	PROCEDURE	NC	R5,F0,F1
CTCE	PROCEDURE	NC	R5,F0,F1
CTCD	PROCEDURE	NC	R5,F0,F1
CTCI	PROCEDURE	NC	R5,F0,F1
CTCK	PROCEDURE	NC	R5,F0,F1
CTCH	PROCEDURE	NC	R5,F0,F1
CTCX	PROCEDURE	NC	R5,F0,F1
CTCO	PROCEDURE	NC	NONE
CTRIMV	PROCEDURE	NC	R2,R3,R4,R5,R6
DACOS	PROCEDURE	NC	R5,F0,F1
DASIN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DACOSH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

DASINH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DATANH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DATAN2	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DATAN	PROCEDURE	NC	F0,F1,F2,F3
DEXP	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DLOG	INTRINSIC	NC	R2,R4,R5,F0,F1
DMAX	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DMDVAL	INTRINSIC	NC	R2,R4,R5,F0,F1
DMIN	INTRINSIC	NC	R4,F0,F1,F2,F3,F4,F5
DMOD	INTRINSIC	NC	R2,R4,R5,F0,F1
DPROD	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DPWRD	PROCEDURE	NC	F0,F1,F2,F3
DPWRI	PROCEDURE	NC	F0,F1,F2,F3
DPWRH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DSIN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DCOS	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DSINH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DCOSH	PROCEDURE	NC	R5
DSLD	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DSQRT	PROCEDURE	NC	NONE
DSST	PROCEDURE	NC	R2,R4,R5,F0,F1
DSUM	INTRINSIC	NC	F0,F1,F2,F3,F4,F5
DTAN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DTANH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
EATAN2	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
ATAN	PROCEDURE	NC	R2,R4,R5,F0,F1
EMAX	INTRINSIC	NC	R2,R4,R5,F0,F1
EMIN	INTRINSIC	NC	R4,F0,F1,F2,F3,F4,F5
EMOD	INTRINSIC	NC	R2,R4,R5,F0,F1
EPROD	INTRINSIC	NC	F0,F1,F2,F3,F4,F5
EPWRE	PROCEDURE	NC	F0,F1,F2,F3
EPWRI	PROCEDURE	NC	F0,F1,F2,F3
EPWRH	PROCEDURE	NC	R2,R4,R5,F0,F1
ESUM	INTRINSIC	NC	F0,F1,F2,F3,F4,F5
ETCC	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
DTCC	PROCEDURE	NC	R4,R5,F0,F1
ETCH	INTRINSIC	YES	R4,R5,F0,F1
DTOH	INTRINSIC	YES	F0,F1,F2,F3
EXP	PROCEDURE	NC	R2,R4,R5,F0,F1
GTBYTE	INTRINSIC	NC	F0,F1
HIN	PROCEDURE	NC	F0,F1
IIN	PROCEDURE	NC	F0,F1
EIN	PROCEDURE	NC	F0,F1
DIN	PROCEDURE	NC	F0,F1
BIN	PROCEDURE	NC	R2,R4,R5,R6
HMAX	INTRINSIC	NC	R2,R4,R5,R6
HMIN	INTRINSIC	NC	R2,R4,R5,R6
HPROD	INTRINSIC	NO	R2,R4,R5,R6
HSUM	INTRINSIC	NC	R2,R4,R5,R6
IMAX	INTRINSIC	NC	R2,R4,R5,R6
IMIN	INTRINSIC	NC	R2,R4,R5,R6,R7
IMOD	INTRINSIC	NC	R2,R4,R5,R6,R7
HMOD	INTRINSIC	NC	F0,F1
ICINIT	PROCEDURE	NC	F0,F1
HCUT	PROCEDURE	NC	

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

CUTER1	PROCEDURE	NC	F0,F1
ICUT	PROCEDURE	NC	F0,F1
ECUT	PROCEDURE	NO	F0,F1
DOUT	PROCEDURE	NC	F0,F1
BOUT	PROCEDURE	NC	F0,F1
SKIP	PROCEDURE	NC	F0,F1
LINE	PROCEDURE	NC	F0,F1
COLUMN	PROCEDURE	NC	F0,F1
TAB	PROCEDURE	NC	F0,F1
PAGE	PROCEDURE	NC	F0,F1
IPROD	INTRINSIC	NO	R2,R4,R5,R6,R7
IPWRI	PROCEDURE	NC	R5
IPWRH	PROCEDURE	NO	R5
HPWRH	PROCEDURE	NC	R5
IREM	INTRINSIC	NC	R2,R4,R5,R6,R7
HREM	INTRINSIC	NC	R2,R4,R5,R6,R7
ISUM	INTRINSIC	NC	R2,R4,R5,R6
ITOC	PROCEDURE	NC	NONE
HTCC	PROCEDURE	NC	NONE
ITOD	INTRINSIC	YES	R4,R5,F0,F1
ITOE	INTRINSIC	YES	R4,R5,F0,F1
KTOC	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1
LCG	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MMRDNP	PROCEDURE	NC	NONE
MMRSNP	PROCEDURE	NC	NONE
MMWDNP	PROCEDURE	NC	F0,F1
MMWSNP	PROCEDURE	NC	F0,F1
MM0DNP	INTRINSIC	NC	R1,R3,R4,R5,R6,R7,F0,F1
MM0SNP	INTRINSIC	NC	R1,R3,R4,R5,R6,R7,F0,F1
MM1DNP	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3
MM1SNP	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1
MM1TNP	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3
MM1WNP	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3
MM11DN	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3
MM11D3	INTRINSIC	NC	R1,R2,R4,R5,F0,F1,F2,F3,F4,F5
MM11SN	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1
MM11S3	INTRINSIC	NC	R1,R2,R4,R5,F0,F1,F2,F3
MM12DN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM12D3	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM12SN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM12S3	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM13DN	INTRINSIC	NC	R2,R4,R5,R6,F0,F1
MM13D3	INTRINSIC	NO	R2,R4,F0,F1
MM13SN	INTRINSIC	NC	R2,R4,R5,R6,F0,F1
MM13S3	INTRINSIC	NO	R2,R4,F0,F1
MM14DN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM14D3	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM14SN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM14S3	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM15DN	INTRINSIC	NO	R1,R4,R5,R6,R7,F0,F1,F2,F3
MM15SN	INTRINSIC	NO	R1,R4,R5,R6,R7,F0,F1,F2,F3
MM17D3	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM17DN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM17S3	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
MM17SN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5

MM6DN	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5
MM6D3	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5
MM6SN	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5
MM6S3	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5
MR0DNP	PROCEDURE	NC	F0,F1
MR0SNP	PROCEDURE	NC	F0,F1
MR1DNP	PROCEDURE	NC	F0,F1
MR1SNP	PROCEDURE	NC	F0,F1
MR1TNP	PROCEDURE	NC	F0,F1
MR1WNP	PROCEDURE	NC	F0,F1
MSTR	PROCEDURE	NO	NONE
MSTRUC	INTRINSIC	NO	R1,R2,R4,R5,R6
MV6DN	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5
MV6D3	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3
MV6SN	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5
MV6S3	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,F0,F1,F2,F3
QSHAPQ	PROCEDURE	NC	F0,F1
RANDOM	PROCEDURE	NC	F0,F1,F2,F3
RANDG	PROCEDURE	NO	F0,F1,F2,F3
RCUND	INTRINSIC	YES	R4,R5,F0,F1
RTCI	INTRINSIC	YES	R4,R5,F0,F1
RTRUNC	INTRINSIC	YES	R4,R5,F0,F1
SFLOOR	INTRINSIC	YES	R4,R5,F0,F1
SCEIL	INTRINSIC	YES	R4,R5,F0,F1
SRCUND	INTRINSIC	YES	R4,R5,F0,F1
STCI	INTRINSIC	NC	R1,R3,R4,F0,F1,F2,F3,F4,F5
SIN	INTRINSIC	NC	R1,R3,R4,F0,F1,F2,F3,F4,F5
SCS	INTRINSIC	NC	F0,F1,F2,F3,F4,F5
SSINH	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
SCOSH	PROCEDURE	NO	R1,R4,R5,R6,R7,F0,F1,F2,F3
SQRT	INTRINSIC	NC	R1,R4,R5,F0,F1
STBYTE	INTRINSIC	NC	F0,F1,F2,F3,F4,F5
STAN	PROCEDURE	NC	F0,F1,F2,F3,F4,F5
STANH	PROCEDURE	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5
VM6DN	INTRINSIC	NO	R1,R2,R3,R4,R5,F0,F1,F2,F3,F4,F5
VM6D3	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5
VM6SN	INTRINSIC	NC	R1,R2,R3,R4,R5,F0,F1,F2,F3
VM6S3	INTRINSIC	NO	R1,R2,R3,R4,R5,R6,R7,F0,F1,F4,F5
VC6DN	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,F0,F1
VC6D3	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,R7,F0,F1,F4,F5
VC6SN	INTRINSIC	NC	R1,R2,R3,R4,R5,R6,F0,F1
VC6S3	INTRINSIC	NO	F0,F1
VR0DN	PROCEDURE	NC	F0,F1
VR0DNP	PROCEDURE	NC	F0,F1
VR0SN	PROCEDURE	NC	F0,F1
VR0SNP	PROCEDURE	NC	F0,F1
VR1DN	PROCEDURE	NC	F0,F1
VR1DNP	PROCEDURE	NC	F0,F1
VR1SN	PROCEDURE	NC	F0,F1
VR1SNP	PROCEDURE	NC	F0,F1
VR1TN	PROCEDURE	NC	F0,F1
VR1TNP	PROCEDURE	NC	F0,F1

```
VR1WN     PROCEDURE     NC          F0,F1
VR1WNP    PROCEDURE     NC          F0,F1
VV0DN     INTRINSIC     NC          R1,R4,R5,F0,F1
VV0DNP    INTRINSIC     NC          R1,R4,R5,R7,F0,F1
VV0SN     INTRINSIC     NC          R1,R4,R5,F0,F1
VV0SNP    INTRINSIC     NC          R1,R4,R5,R7,F0,F1
VV1DN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1
VV1D3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3,F4,F5
VV1D3P    INTRINSIC     NO          R1,R2,R4,R5,R6,R7,F0,F1
VV1DNP    INTRINSIC     NC          R1,R2,R4,R5,R6,R7,F0,F1
VV1SN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1
VV1S3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3,F4,F5
VV1S3P    INTRINSIC     NC          R1,R2,R4,R5,R6,R7,F0,F1
VV1SNP    INTRINSIC     NO          R1,R2,R4,R5,R6,R7,F0,F1
VV1TN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1
VV1T3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3,F4,F5
VV1T3P    INTRINSIC     NC          R1,R2,R4,R5,R6,R7,F0,F1
VV1TNP    INTRINSIC     NO          R1,R2,R4,R5,R6,R7,F0,F1
VV1WN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1
VV1W3     INTRINSIC     NC          R1,R2,R4,F0,F1
VV1W3P    INTRINSIC     NC          R1,R2,R4,R5,R6,R7,F0,F1
VV1WNP    INTRINSIC     NC          R1,R2,R4,R5,R6,R7,F0,F1
VV10D3    PROCEDURE     NC          F0,F1,F2,F3,F4,F5
VV10DN    PROCEDURE     NC          F0,F1,F2,F3,F4,F5
VV9D3     PROCEDURE     NO          F0,F1,F2,F3,F4,F5
VV9DN     PROCEDURE     NC          F0,F1,F2,F3,F4,F5
VV10S3    PROCEDURE     NC          F0,F1,F2,F3,F4,F5
VV10SN    PROCEDURE     NC          F0,F1,F2,F3,F4,F5
VV9S3     PROCEDURE     NC          F0,F1,F2,F3,F4,F5
VV9SN     PROCEDURE     NC          F0,F1,F2,F3,F4,F5
VV2DN     INTRINSIC     NC          R1,R2,R3,R4,R5,F0,F1
VV2D3     INTRINSIC     NC          R1,R2,R3,R4,F0,F1,F2,F3,F4,F5
VV2SN     INTRINSIC     NC          R1,R2,R3,R4,R5,F0,F1
VV2S3     INTRINSIC     NC          R1,R2,R3,R4,F0,F1,F2,F3,F4,F5
VV3DN     INTRINSIC     NO          R1,R2,R3,R4,R5,F0,F1
VV3D3     INTRINSIC     NC          R1,R2,R3,R4,F0,F1
VV3SN     INTRINSIC     NO          R1,R2,R3,R4,R5,F0,F1
VV3S3     INTRINSIC     NC          R1,R2,R3,R4,F0,F1,F2,F3,F4,F5
VV4DN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1,F2,F3
VV4D3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3
VV4SN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1,F2,F3
VV4S3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3
VV5DN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1,F2,F3
VV5D3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3,F4,F5
VV5SN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1,F2,F3
VV5S3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3
VV6DN     INTRINSIC     NC          R1,R2,R3,R4,R5,F0,F1,F2,F3
VV6D3     INTRINSIC     NC          R2,R3,R4,F0,F1,F2,F3
VV6SN     INTRINSIC     NO          R1,R2,R3,R4,R5,F0,F1,F2,F3
VV6S3     INTRINSIC     NC          R2,R3,R4,F0,F1,F2,F3
VV7DN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1
VV7D3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3,F4,F5
VV7SN     INTRINSIC     NC          R1,R2,R4,R5,F0,F1
VV7S3     INTRINSIC     NC          R1,R2,R4,F0,F1,F2,F3,F4,F5
VV8D3     INTRINSIC     NC          R1,R2,R3,R4,R5,F0,F1
```



```
VV8DN     INTRINSIC     NC          R1,R2,R3,R4,R5,F0,F1
VV8S3     INTRINSIC     NC          R1,R2,R3,R4,R5,F0,F1
VV8SN     INTRINSIC     NC          R1,R2,R3,R4,R5,F0,F1
VX6D3     INTRINSIC     NC          R1,R2,R3,R4,F0,F1,F2,F3,F4,F5
VX6S3     INTRINSIC     NC          R1,R2,R3,R4,F0,F1,F2,F3
XTCC      INTRINSIC     NC          R1,R2,R3,R4,R5,R6,R7,F0,F1
CTCC      INTRINSIC     NC          R1,R2,R3,R4,R5,R6,R7,F0,F1
```

5.2.5 Referencing Conventions

5.2.5.1 CSECT Names. In order to comply with the CSECT naming standards described in the HAL/SDL ICD, all library code CSECTs begin with two alphabetic characters (A-Z)*. All library primary names and aliases are unique to 6 characters.

Whenever a data CSECT is needed for a particular library module, it is given the CSECT name #Lnnnnnn, where nnnnnn is the first 6 characters of the primary entry name.

5.2.5.2 ZCON's. For each primary entry point and alternate entry point in the runtime library, a member exists in a separate ZCON library. The members in the ZCON library contain address constants which refer to the actual entry points. Thus, for the library routine named SIN which has an entry point named COS, there are two members in the ZCON library named #QSIN and #QCOS. These #Q modules contain references to the respective entry points. The individual ZCONs in the ZCON library are created by assembly code like the following:

```
#QSIN CSECT
      DC Z(SIN,,X'E')
      EXTRN SIN
      END
```

Some library routines make reference to other library routines via the ACALL macro (see Section 5.2.6). The ACALL macro does not make reference via a ZCON as is done when compiler-emitted code references a library routine. Use of the ACALL macro for inter-library routine reference means that a referenced routine must be in the same machine sector as the referencing routine, or must be in sector zero.

* Sector 0 routines are an exception: their CSECT names begin with #0. This is to conform to link editor conventions for routines which must reside in sector 0. Sector 0 routines are identified in the list in Section 5.2.4.3 and in the boxed area of the individual library routine description.

5.2.6 Coding Structure

The following outline represents the standard coding structure of all library members.

1 TITLE

2 WORKAREA macro definition

used only if addition stack storage was needed

3 AMAIN

4 * Comment card describing the function of the primary entry point

5 INPUT

6 OUTPUT

7 body of executable code including use of WORK, AERROR, AEXIT macros where needed and alternate entry points defined using the AENTRY macro, function comment card, and INPUT and OUTPUT macros in the same manner as the primary entry point.

8 DC constant area addressed via PC relative mode

9 ADATA, followed by a DC constant area addressed via base and displacement mode.

used only if constants need to be indexed

10 ACLOSE

5.2.7 The Macro Library

To standardize interface conventions, automate production of commonly used code sequences, and impose a structure to the runtime library, a series of macros are used. This section describes the function, use, and expansion of these macros. Lower case letters are used to indicate variable fields. Square brackets [ ] indicate optional fields, braces { } indicate a choice of required fields. Complete listings of all the macro source code is also included.

- AMAIN

```
name AMAIN   [INTSIC = {YES      }]
             [       {INTERNAL}]
             [ACALL  = YES      ]
             [SECTOR = 0        ]
```

Function:

Defines "name" as the primary entry point of a routine.

INTSIC=YES:

Defines the routine (and any entry points) as an intrinsic. If the INTSIC operand is omitted, the routine is defined as a procedure.

INTSIC=INTERNAL:

Defines an intrinsic which is called only by other routines in the library. At present, this is only GTBYTE and STBYTE.

ACALL=YES:

(Valid only for procedure routines.) Allows use of the ACALL macro within the routine (See ACALL description).

SECTOR=0:

Defines the routine (intrinsic or procedure) as a Sector 0 routine.

Expansion:

The macro first defines the primary entry "name" (the AMAIN label) as the CSECT name, unless SECTOR=0 was specified. In the latter case, the CSECT name is generated by prefixing "name" with #0, and the primary entry "name" is defined using the DS and ENTRY statements. The options selected via the AMAIN operands are saved in global SETB variables for testing by the other macros. If either INTSIC option was selected, the macro ends. Otherwise, a procedure is being defined, so the STACK DSECT is generated.

The STACK DSECT consists of a standard 18 halfword area, including symbols for the saved copies of the fixed point register parameters (ARG2, ARG4, ARG5, ARG6, ARG7), followed by the WORKAREA macro. The WORKAREA macro is the means by which additional storage beyond the standard stack of 18 halfwords may be defined. If such storage is needed a local WORKAREA macro must have been defined earlier in the source which contains the appropriate DS assembler statements. These statements are thus incorporated as the remainder of the STACK DSECT. If additional storage is not needed, the local WORKAREA macro is not defined. As a result, the system WORKAREA macro is invoked, which does not define any storage, leaving the STACK DSECT at its standard length. The system WORKAREA macro also sets a global SETB variable, which is tested later by the AMAIN macro to determine if the stack is standard or augmented. The STACK DSECT is then terminated by resuming the original CSECT. The STACK DSECT is defined in this sequence so that the assembler will output the SYM records in the order expected by the link editor's stack size algorithm. A USING statement is generated to give addressibility to the stack area. Finally, the executable code of the entry prologue is generated. This consists of an NIST instruction to zero the 10th halfword of the new stack frame, establishing a null ON ERROR environment. In addition, if both ACALL=YES is specified and a local WORKAREA provided, the default stack size of 18 set up by the SCAL microcode will be insufficient, so an IAL to set up the new stack size is generated.

- AENTRY

name AENTRY

Function:

Defines "name" as a secondary entry point.

Expansion:

"name" is externally defined using the DS and ENTRY statements. If the routine was defined as an intrinsic, the macro ends. Otherwise, the executable code of the entry prologue is generated in the same manner as the AMAIN macro.

- AEXIT

AEXIT [CC = (KEEP / (rx) / EQ / NE)] [COND=code]

Function:

Cause return of control from a procedure or intrinsic routine.

CC:

Used to pass a condition code back to the caller. It can be used only if OUTPUT CC was specified. (See OUTPUT macro.)

Valid for Intrinsics Only:

CC=KEEP:

Passes back the condition code as is.

CC=(rx):

Passes back the condition code generated by a LR rx, rx.

Valid for Procedures Only:

CC=EQ:

Passes back an equal (B'00') condition code.

CC=NE:

Passes back a not equal (B'11') condition code.

Note: The CC= operand is used in the following 8 routines: CPR, CPRA, CTSR, CSTRUCT, VV8DN, VV8D3, VV8SN, and VV8S3.

COND=code:

Used to do a conditional return, i.e. based on the current condition code. Valid for procedures only. "code" is either a number used as the mask on a BC opcode, or a letter or letter pair representing the mask in the extended BC mnemonic op codes. (E, Z, NE, NZ, H, O, L, M, HE, LE, NL, NM, NH, NO). This operand may be used to improve the efficiency of some routines. If used, be sure valid executable code follows it, so the fall through case is valid.

Expansion:

The code generated by the AEXIT macro depends primarily on whether the routine is an intrinsic or procedure, and secondarily on what operands were supplied, and, in the case of intrinsics, what fixed point registers were used. The expansions for intrinsics and procedures are described separately.

Intrinsics:

If register(s) R1 and/or R3 have been defined (see INPUT, OUTPUT, and WORK macros), it is assumed they have been modified and must be restored from the stack, since they are the addressing registers for compiled code. This is done via the appropriate LH instruction(s), or IHL and SLL instructions if CC=KEEP was specified, since LH would destroy the existing condition code. If CC=(rx) was specified, a LR rx,rx is generated to set the condition code. Finally, a BCRE or BCR is generated to cause a return to the caller. A BCR is generated if SECTOR=0 or INTSIC=INTERNAL was specified on the AMAIN macro.

Procedures:

If CC=EQ or CC=NE was specified, the condition code bits in the return PSW in the stack are zeroed or set via the ZB or SB instruction. Then, an SRET instruction is generated with a mask of 7 if the COND operand was omitted, or the appropriate mask if it was supplied.

- INPUT

```
INPUT  { register spec   type comments }
       { NONE                          }
```

Function:

Defines input interface of primary or alternate entry point and symbolic names for the register(s).

Register Spec:

One of R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5, F6, or F7. If there is no input (RANDOM, RANG only), code NONE. If there is more than one, use continuation lines for each subsequent one (see examples).

Type Comments:

type	precision	units
SCALAR	SINGLE/DOUBLE	RADIANS
MATRIX(3,3)		
MATRIX(N,N)		
VECTOR(3)		
VECTOR(N)		
INTEGER(N)		
CHARACTER		

Examples:

```
                 col. 16                                        col. 72
                   ↓                                              ↓
(1)   INPUT      F0     SCALAR       SINGLE        RADIANS

(2)   INPUT      R2,    VECTOR(N)    DOUBLE                        X
                 R3,    VECTOR(N)    DOUBLE                        X
                 R5     INTEGER(N)   SINGLE
```

Note: R1 and R3 are illegal inputs for procedure routines, and R4 is illegal for intrinsic routines.

Expansion:

For each register spec supplied, the macro checks for a valid register symbolic, or for the special case of NONE. If the symbolic register name has not been previously defined, an EQU statement is generated to define it. The macro also tests for the illegal use of R1 or R3 for a procedure parameter and R4 for an intrinsic. A global arrayed SETB variable is set, which in conjunction with the AMAIN, AENTRY, and ACLOSE macros, will guarantee that an INPUT macro has been supplied for each entry point (see ACLOSE macro).

- OUTPUT

```
OUTPUT  { register spec   type comments }
        { NONE                          }
        { CC                            }
```

Function:

Defines output interface of primary or alternate entry point.

Operand form is identical to that of INPUT macro, with the addition of CC as a possibility. This indicates that the condition code is the output of the routine. If CC is specified, the CC= option of the AEXIT macro must be used.

Expansion:

Same as for INPUT macro, except for special processing for the CC operand. If CC is supplied, a global SETB variable is set which is tested by the AEXIT macro for consistency with its CC operand.

- WORK

```
WORK  {register spec}
```

Function:

Defines work registers.

Expansion:

Similar to INPUT and OUTPUT, except that this macro is required only if additional register symbols need to be defined.

- ABAL

```
ABAL  name
```

Function:

Calls the intrinsic routine "name", valid only in a procedure routine.

Expansion:

Generates a BAL 4, name, and an EXTRN statement if "name" has not been previously defined.

- ACALL

```
ACALL name
```

Function:

Calls the procedure routine "name", valid only in a procedure routine defined with ACALL=YES option.

Expansion:

Generates an SCAL 0, name, and an EXTRN statement if "name" has not been previously defined.

- AERROR

```
AERROR number   cause comment
```

Function:

Generates a send error SVC instruction to signal a run time error to the FCOS.

Number:

The error number.

Cause Comment:

Brief description of the cause of the error.

Expansion:

This macro accumulates, in GBLA variables, all errors sent within one assembly. It also checks to see that the error number indicates as an argument to AERROR is less than a maximum value. The actual code emitted is an SVC in which the operand is the label of an SVC parameter list to be emitted by the ADATA or ACLOSE macro via the ERRPARMS macro. If any error is sent more than once in an assembly, AERROR insures that only one SVC parameter list for that error is used.

- ADATA

```
ADATA
```

Function:

Defines the start of a separate data CSECT for indexable constant data.

Expansion:

A CSECT is created with the name #Lnnnnnn where nnnnnn is the first 6 characters of the primary CSECT name defined by the AMAIN macro. The ADATA macro ends leaving the data CSECT in effect so that any user-defined data following the macro call will be part of the data CSECT. The ERRPARMS macro is invoked so that any possible AERROR SVC parameter lists will appear before the indexed data. This is necessary so that the assembler will use the direct addressing mode instead of base and displacement.

- ACLOSE

  ACLOSE

  Function:

  Terminates the assembly.

  Expansion:

  The macro first invokes the ERRPARMS macro to create the AERROR SVC parameter lists. (See ERRPARMS macro.) It then checks via arrayed global SETB variables if INPUT and OUTPUT macros were supplied for each entry point. Finally, it generates an END assembler statement, terminating the assembly.

- ERRPARMS

  ERRPARMS

  Function:

  Generates SVC parameter lists for the AERROR macro.

  Expansion:

  This macro is invoked by the ADATA and ACLOSE macro. It first tests a global SETB variable to see if it has already been invoked, in which case the macro does nothing. Otherwise, it generates a CSECT statement to define the data CSECT (FCOS parameter lists must reside in the data sector). The CSECT name is #Lname, where "name" is the primary entry name. The parameter lists are generated by looping through arrayed global SETA variables in which the AERROR macro saved the unique error numbers. ERRPARMS is invoked by the ADATA macro because the parameter lists must be before any indexed data following the optional ADATA macro. It is invoked by the ACLOSE macro in case the ADATA macro is not used.

- WORKAREA

  WORKAREA

  Function:

  An automatically invoked, user-created macro used to define extensions of the stack area for temporary reentrant storage. The WORKAREA macro is invoked by the AMAIN macro in procedure routines. A system supplied default is invoked in the absence of a user-created macro.

  Expansion:

  The system WORKAREA macro merely sets a global SETB variable which is tested by the AMAIN macro to determine whether the system or user macro is being expanded.

- AMAIN

```
         MACRO                                                          00000100
&NAME    AMAIN &INTSIC=NO,&ACALL=NO,&SECTOR=                            00000200
         GBLA  &ENTCNT                                                  00000300
         GBLB  &CALLOK,&LIB,&NOEXTRA,&INTERN,&SECT0                     00000400
         GBLC  &CSECT,&NAMES(20)                                        00000500
&CSECT   SETC  '&NAME'                                                  00000600
&ENTCNT  SETA  &ENTCNT+1                                                00000700
&NAMES(&ENTCNT) SETC '&NAME'                                            00000800
*********************************************************************** 00000900
*                                                                       00001000
*        PRIMARY ENTRY POINT                                            00001100
*                                                                       00001200
*********************************************************************** 00001300
&CNAME   SETC  '&NAME'                                                  00001400
         AIF   ('&SECTOR' EQ '').REG                                    00001500
         AIF   ('&SECTOR' NE '0').BADSECT                               00001600
&CNAME   SETC  '#0'.'&NAME'                                             00001700
&SECT0   SETB  1                                                        00001800
&CNAME   CSECT                                                          00001900
&NAME    DS    0H              PRIMARY ENTRY POINT                      00002000
         ENTRY &NAME                                                    00002100
         AGO   .COMM                                                    00002200
.REG     ANOP                                                           00002300
&CNAME   CSECT                                                          00002400
.COMM    ANOP                                                           00002500
&LIB     SETB  ('&INTSIC' EQ 'NO')                                      00002600
&INTERN  SETB  ('&INTSIC' EQ 'INTERNAL')                                00002700
         AIF   (NOT &LIB).SPACE                                         00002800
STACK    DSECT                                                          00002900
*        DS    18H             STANDARD STACK AREA DEFINITION           00003000
         DS    F               PSW (LEFT HALF)                          00003100
         DS    2F              R0,R1                                    00003200
ARG2     DS    F               R2                                       00003300
         DS    F               R3                                       00003400
ARG4     DS    F               R4                                       00003500
ARG5     DS    F               R5                                       00003600
ARG6     DS    F               R6                                       00003700
ARG7     DS    F               R7                                       00003800
*        END OF STANDARD STACK AREA                                     00003900
         WORKAREA                                                       00004000
STACKEND DS    0F              END OF COMBINED STACK AREA               00004100
&CNAME   CSECT                                                          00004200
         USING STACK,0         ADDRESS STACK AREA                       00004300
&CALLCK  SETB  ('&ACALL' EQ 'YES')                                      00004400
         AIF   (&NOEXTRA OR NOT &CALLOK).NIST                           00004500
         IAL   0,STACKEND-STACK SET STACK SIZE                          00004600
.NIST    NIST  9(0),0          CLEAR ON ERROR INFO (LCL DATA PTR)       00004700
.SPACE   SPACE                                                          00004800
         MEXIT                                                          00004900
.BADSECT MNOTE 4,'ONLY SECTOR=0 MAY BE SPECIFIED'                       00005000
         MEND                                                           00005100
```


- AENTRY

```
         MACRO                                                          00000100
&NAME    AENTRY                                                         00000200
         GBLA  &ENTCNT                                                  00000300
         GBLB  &CALLOK,&NOEXTRA,&LIB                                    00000400
         GBLC  &NAMES(20)                                               00000500
*********************************************************************** 00000600
*                                                                       00000700
*        SECONDARY ENTRY POINT                                          00000800
*                                                                       00000900
*********************************************************************** 00001000
&ENTCNT  SETA  &ENTCNT+1                                                00001100
&NAMES(&ENTCNT) SETC '&NAME'                                            00001200
&NAME    DS    0H                                                       00001300
         ENTRY &NAME                                                    00001400
         AIF   (NOT &LIB).SPACE                                         00001500
         AIF   (&NOEXTRA OR NOT &CALLOK).NIST                           00001600
         IAL   0,STACKEND-STACK SET STACK SIZE                          00001700
.NIST    NIST  9(0),0          CLEAR ERROR VECTOR POINTER               00001800
.SPACE   SPACE                                                          00001900
         MEND                                                           00002000
```

- AEXIT

```
         MACRO                                                          00000100
&NAME    AEXIT &CC=,&COND=                                              00000200
         GBLA  &RET                                                     00000300
         GBLB  &LIB,&CCTYPE,&INTERN,&SECT0                              00000400
         LCLA  &MASK                                                    00000500
&MASK    SETA  7                                                        00000600
&RET     SETA  &RET+1                                                   00000700
*********RETURN TO CALLER*************************************************00000800
&NAME    DS    0H                                                       00000900
         AIF   ('&CC' EQ '' AND NOT &CCTYPE).OK1                        00001000
         AIF   ('&CC' NE '' AND &CCTYPE).OK1                            00001100
         MNOTE 1,'CONFLICTING CC OPERANDS IN OUTPUT AND AEXIT MACROS'   00001200
.OK1     AIF   (&LIB).LIB                                               00001300
.*       GENERATE EXIT SEQUENCE FOR INTRINSICS                          00001400
         AIF   ('&COND' EQ '').OK2                                      00001500
         MNOTE 4,'COND OPERAND INVALID FOR INTRINSIC'                   00001600
.OK2     AIF   ('&CC' EQ '').LHS                                        00001700
         AIF   ('&CC'(1,2) EQ '(R').LHS                                 00001800
         AIF   ('&CC' EQ 'KEEP').IHLS                                   00001900
         MNOTE 4,'INVALID CC OPERAND FOR INTRINSIC'                     00002000
.LHS     AIF   (&INTERN).SKIP1                                          00002100
         AIF   (NOT D'R3).SKIP3                                         00002200
         LH    3,9(0)          RESTORE LOCAL DATA BASE                  00002300
.SKIP3   AIF   (NOT D'R1).SKIP1                                         00002400
         LH    1,5(0)          RESTORE PROGRAM DATA BASE                00002500
.SKIP1   AIF   ('&CC' EQ '').BCRE                                       00002600
&R       SETC  '&CC(1)'                                                 00002700
         AIF   ('&R' NE 'R1' AND '&R' NE 'R3').LROK                     00002800
         MNOTE 4,'INVALID REGISTER IN CC= OPERAND'                      00002900
.LROK    LR    &R,&R           SET CONDITION CODE                       00003000
         AGO   .BCRE                                                    00003100
```

• AEXIT (CONTINUED)

```
.IHLS    AIF   (&INTERN).BCRE                                          00003200
         AIF   (NOT D'R3).SKIPR3                                       00003300
         IHL   R3,9(0)          LOAD R3, PRESERVING CC                 00003400
         SLL   R3,16            POSITION IN UPPER HALFWORD             00003500
.SKIPR3  AIF   (NOT D'R1).BCRE                                         00003600
         IHL   R1,9(0)          LOAD R1, PRESERVING CC                 00003700
         SLL   R1,16            POSITION IN UPPER HALFWORD             00003800
.BCRE    ANOP                                                          00003900
         AIF   (&SECTO OR &INTERN).BCR                                 00004000
$RET&RET BCRE  7,4              RETURN TO CALLER                       00004100
*********************************************************************  00004200
         SPACE                                                         00004300
         MEXIT                                                         00004400
.BCR     ANOP                                                          00004500
$RET&RET BCR   7,4              RETURN TO CALLER                       00004600
*********************************************************************  00004700
         SPACE                                                         00004800
         MEXIT                                                         00004900
.LIB     AIF   ('&CC' EQ '' OR '&COND' EQ '').OK3                      00005000
         MNOTE 4,'CC AND COND OPERANDS ARE MUTUALLY EXCLUSIVE'         00005100
.OK3     AIF   ('&CC' EQ '').NOCC                                      00005200
         AIF   ('&CC' EQ 'EQ').ZB                                      00005300
         AIF   ('&CC' EQ 'NE').SB                                      00005400
         MNOTE 4,'INVALID CC OPERAND FOR PROCEDURE ROUTINE'            00005500
.ZB      ZB    1(0),X'C000'     SET PSW CC TO 00 (EQ)                  00005600
         AGO   .NOCC                                                   00005700
.SB      SB    1(0),X'C000'     SET PSW CC TO 11 (LT (NE))             00005800
.NOCC    AIF   (T'&COND NE 'N').CONDTST                                00005900
&MASK    SETA  &COND                                                   00006000
         AGO   .SRET                                                   00006100
.CONDTST AIF   ('&COND' EQ '').SRET                                    00006200
&MASK    SETA  1                                                       00006300
         AIF   ('&COND' EQ 'H' OR '&COND' EQ 'O').SRET                 00006400
&MASK    SETA  2                                                       00006500
         AIF   ('&COND' EQ 'L' OR '&COND' EQ 'M').SRET                 00006600
&MASK    SETA  3                                                       00006700
         AIF   ('&COND' EQ 'NE' OR '&COND' EQ 'NZ').SRET               00006800
&MASK    SETA  4                                                       00006900
         AIF   ('&COND' EQ 'E' OR '&COND' EQ 'Z').SRET                 00007000
&MASK    SETA  5                                                       00007100
    AIF ('&CCND' EQ 'HE' OR '&COND' EQ 'NL' OR '&COND' EQ 'NM').SRET   00007200
&MASK    SETA  6                                                       00007300
    AIF ('&CCND' EQ 'LE' OR '&COND' EQ 'NH' OR '&COND' EQ 'NO').SRET   00007400
&MASK    SETA  7                                                       00007500
         MNOTE 4,'INVALID COND OPERAND'                                00007600
.SRET    ANOP                                                          00007700
$RET&RET SRET  &MASK,0          RETURN TO CALLER                       00007800
*********************************************************************  00007900
         SPACE                                                         00008000
         MEND                                                          00008100
```


• INPUT

```
         MACRO                                                         00000100
         INPUT &X                                                      00000200
         GBLA  &ENTCNT                                                 00000300
         GBLB  &INPUT(20),&LIB                                         00000400
         AIF   (N'&SYSLIST EQ 0).EMPTY                                 00000500
&INPUT(&ENTCNT) SETB 1                                                 00000600
         AIF   ('&X' EQ 'NONE').SPACE                                  00000700
&I       SETA  1                                                       00000800
&LAST    SETA  N'&SYSLIST                                              00000900
.LOOP    AIF   (K'&SYSLIST(&I) NE 2).BADREG                            00001000
&R       SETC  '&SYSLIST(&I)'                                          00001100
         AIF   ('&R'(1,1) NE 'R' AND '&R'(1,1) NE 'R').BADREG          00001200
         AIF   ('&R' EQ 'R0').BADREG                                   00001300
         AIF   (&LIB AND ('&R' EQ 'R1' OR '&R' EQ 'R3')).INVREG1       00001400
         AIF   (NOT &LIB AND '&R' EQ 'R4').INVREG2                     00001500
         AIF   (D'&R).NEXT                                             00001600
&N       SETC  '&R'(2,1)                                               00001700
&R       EQU   &N                                                      00001800
.NEXT    ANOP                                                          00001900
&I       SETA  &I+1                                                    00002000
         AIF   (&I LE &LAST).LOOP                                      00002100
.SPACE   SPACE                                                         00002110
         MEXIT                                                         00002200
.BADREG  MNOTE 4,' ILLEGAL REGISTER SPECIFICATION - &SYSLIST(&I)'      00002300
         AGO   .NEXT                                                   00002400
.INVREG1 MNOTE 4,'&R INVALID INPUT FOR PROCEDURE ROUTINE'              00002500
         AGO   .NEXT                                                   00002600
.INVREG2 MNOTE 4,'R4 INVALID INPUT FOR INTRINSIC'                      00002700
         AGO   .NEXT                                                   00002800
.EMPTY   MNOTE 4,'OPERAND REQUIRED'                                    00002900
         MEND                                                          00003000
```

- OUTPUT

```
         MACRO                                                          00000100
         OUTPUT &X                                                      00000200
         GBLA  &ENTCNT                                                  00000300
         GBLB  &OUTPUT(20),&CCTYPE,&LIB                                 00000400
         AIF   (N'&SYSLIST EQ 0).EMPTY                                  00000500
&OUTPUT(&ENTCNT) SETB 1                                                 00000600
         AIF   ('&X' EQ 'NONE').SPACE                                   00000700
&I       SETA  1                                                        00000800
&LAST    SETA  N'&SYSLIST                                               00000900
.LOOP    AIF   (K'&SYSLIST(&I) NE 2).BADREG                             00001000
&R       SETC  '&SYSLIST(&I)'                                           00001100
         AIF   ('&R' EQ 'CC').CCTYPE                                    00001200
         AIF   ('&R'(1,1) NE 'F' AND '&R'(1,1) NE 'R').BADREG           00001300
         AIF   ('&R' EQ 'R0').BADREG                                    00001400
         AIF   (&LIB AND ('&R' EQ 'R1' OR '&R' EQ 'R3')).INVREG1        00001500
         AIF   (NOT &LIB AND '&R' EQ 'R4').INVREG2                      00001600
         AIF   (D'&R).NEXT                                              00001700
&N       SETC  '&R'(2,1)                                                00001800
&R       EQU   &N                                                       00001900
.NEXT    ANOP                                                           00002000
&I       SETA  &I+1                                                     00002100
         AIF   (&I LE &LAST).LOOP                                       00002200
.SPACE   SPACE                                                          00002210
         MEXIT                                                          00002300
.BADREG  MNOTE 4,' ILLEGAL REGISTER SPECIFICATION - &SYSLIST(&I)'       00002400
         AGO   .NEXT                                                    00002500
.CCTYPE  ANOP                                                           00002600
&CCTYPE  SETB  1                                                        00002700
         AGO   .NEXT                                                    00002800
.INVREG1 MNOTE 4,'&R INVALID OUTPUT FOR PROCEDURE ROUTINE'              00002900
         AGO   .NEXT                                                    00003000
.INVREG2 MNOTE 4,'R4 INVALID OUTPUT FOR INTRINSIC'                      00003100
         AGO   .NEXT                                                    00003200
.EMPTY   MNOTE 4,'OPERAND REQUIRED'                                     00003300
         MEND                                                           00003400
```


- WORK

```
         MACRO                                                          00000100
         WORK  &X                                                       00000200
         GBLB  &LIB,&NCEXTRA                                            00000300
         AIF   ('&X' EQ 'NONE').SPACE                                   00000400
&I       SETA  1                                                        00000500
&LAST    SETA  N'&SYSLIST                                               00000600
.LOOP    AIF   (K'&SYSLIST(&I) NE 2).BADREG                             00000700
&R       SETC  '&SYSLIST(&I)'                                           00000800
         AIF   ('&R'(1,1) NE 'F' AND '&R'(1,1) NE 'R').BADREG           00000900
         AIF   ('&R' EQ 'R0').BADREG                                    00001000
         AIF   ('&R' NE 'F6').TESTD                                     00001100
 MNOTE '***** WARNING: F6 MUST BE PRESERVED ACROSS CALLS'               00001200
.TESTD   AIF   (D'&R).NEXT                                              00001300
&N       SETC  '&R'(2,1)                                                00001400
&R       EQU   &N                                                       00001500
.NEXT    ANOP                                                           00001600
&I       SETA  &I+1                                                     00001700
         AIF   (&I LE &LAST).LOOP                                       00001800
.SPACE   SPACE                                                          00001900
         MEXIT                                                          00002000
.BADREG  MNOTE 4,' ILLEGAL REGISTER SPECIFICATION - &SYSLIST(&I)'       00002100
         AGO   .NEXT                                                    00002200
         MEND                                                           00002300
```

- ABAL

```
         MACRO                                                          00000100
&NAME    ABAL  &P                                                       00000200
         GBLB  &LIB                                                     00000250
         AIF   (&LIB).OK                                                00000300
         MNOTE 4,'ABAL MACRO ILLEGAL FROM INTRINSIC'                    00000400
         MEXIT                                                          00000500
.OK      AIF   (D'&P).SKIP                                              00000600
         EXTEN &P                                                       00000700
.SKIP    ANOP                                                           00000800
&NAME    BAL   4,&P            CALL INTRINSIC                           00000900
         MEND                                                           00001000
```

- ACALL

```
         MACRO                                                          00000100
&NAME    ACALL &P                                                       00000200
         GBLB  &CALLOK                                                  00000300
         AIF   (&CALLOK).CALL                                           00000400
         MNOTE 12,'ACALL OPTION NOT SPECIFIED IN AMAIN OR INTSIC=YES SPX00000500
               ECIFIED'                                                 00000600
         MEXIT                                                          00000700
.CALL    AIF   (D'&P).SKIP                                              00000800
         EXTRN &P                                                       00000900
.SKIP    ANOP                                                           00001000
&NAME    SCAL  0,&P          CALL PROCEDURE ROUTINE                     00001100
         MEND                                                           00001200
```

- AERROR

```
          MACRO                                                         00000100
&NAME     AERROR &NUM,&GROUP=4                                          00000200
          GBLA  &ERRCNT,&ERRNUMS(10),&ERRGRPS(10)                       00000300
          LCLA  &I                                                      00000400
          AIF   (&NUM GT 62).BADNUM                                     00000500
&I        SETA  &ERRCNT                                                 00000600
.DUPLOOP  AIF   (&I LE 0).NEWERR                                        00000700
          AIF   (&NUM EQ &ERRNUMS(&I) AND &GROUP EQ &ERRGRPS(&I)).DUP   00000800
&I        SETA  &I-1                                                    00000900
          AGO   .DUPLOOP                                                00001000
.NEWERR   ANOP                                                          00001100
&ERRCNT   SETA  &ERRCNT+1                                               00001200
&I        SETA  &ERRCNT                                                 00001300
&ERRNUMS(&I) SETA &NUM                                                  00001400
&ERRGRPS(&I) SETA &GROUP                                                00001500
.DUP      ANOP                                                          00001600
*********ISSUE SEND ERROR SVC*****************************************  00001700
&NAME     SVC   AERROR&I       ISSUE SEND ERROR SVC                     00001800
*********SEND ERROR SVC RETURNS CONTROL FOR STANDARD FIXUP************  00001900
          MEXIT                                                         00002000
.BADNUM   MNOTE 12,'ERROR NUMBER GREATER THAN 62'                       00002100
          MEND                                                          00002200
```

- ADATA

```
         MACRO                                                          00000100
         ADATA                                                          00000200
         GBLC  &CSECT                                                   00000300
*********DATA CSECT***************************************************  00000400
         ERRPARMS                                                       00000500
&DCSECT  SETC  '#L'.'&CSECT'(1,6)                                       00000600
&DCSECT  CSECT                                                          00000700
         MEND                                                           00000800
```


- ACLOSE

```
         MACRO                                                          00000100
         ACLOSZ                                                         00000200
         GBLA  &ENTCNT                                                  00000300
         GBLB  &INPUT(20),&OUTPUT(20)                                   00000400
         GBLC  &NAMES(20)                                               00000500
         ERRPARMS                                                       00000600
&I       SETA  1                                                        00000700
.LOOP    AIF   (&INPUT(&I)).INOK                                        00000800
         MNOTE 1,'INPUT NOT SPECIFIED FOR &NAMES(&I)'                   00000900
.INOK    AIF   (&OUTPUT(&I)).OUTOK                                      00001000
         MNOTE 1,'OUTPUT NOT SPECIFIED FOR &NAMES(&I)'                  00001100
.OUTOK   ANOP                                                           00001200
&I       SETA  &I+1                                                     00001300
         AIF   (&I LE &ENTCNT).LOOP                                     00001400
         END                                                            00001500
         MEND                                                           00001600
```

- ERRPARMS

```
         MACRO                                                          00000100
         ERRPARMS                                                       00000200
         GBLA  &ERRCNT,&ERRNUMS(10),&ERRGRPS(10)                        00000300
         GBLB  &DONE                                                    00000400
         GBLC  &CSECT                                                   00000500
         LCLA  &I                                                       00000600
         LCLC  &S                                                       00000700
         AIF   (&DONE).MEND                                             00000800
&DONE    SETB  1                                                        00000900
         LTORG                                                          00001000
**********ERROR PARAMETER AREA*************************************    00001100
         AIF   (&ERRCNT EQ 0).NOERROR                                   00001200
&ECSECT  SETC  '#L'.'&CSECT'(1,6)                                       00001300
&ECSECT  CSECT                                                          00001400
         AIF   (&ERRCNT EQ 1).MSG                                       00001500
&S       SETC  'S'                                                      00001600
.MSG     MNOTE '***  &CSECT SENDS THE FOLLOWING ERROR&S'                00001700
.LOOP    ANOP                                                           00001800
&I       SETA  &I+1                                                     00001900
         SPACE 2                                                        00002000
         MNOTE '***  ERROR NUMBER &ERRNUMS(&I) IN GROUP &ERRGRPS(&I)'  00002100
         SPACE 1                                                        00002200
AERROR&I DC    H'20'          SVC CODE FOR SEND ERROR                  00002300
         DC    Y(&ERRGRPS(&I)*256+&ERRNUMS(&I)) 8 BIT GROUP AND NUMBER  00002400
         AIF   (&I LT &ERRCNT).LOOP                                     00002500
         AGO   .COMMON                                                  00002600
.NOERROR MNOTE '***  NO ERRORS SENT IN &CSECT'                          00002700
.COMMON  ANOP                                                           00002800
**********END OF ERROR PARAMETER AREA*********************************  00002900
.MEND    MEND                                                           00003000
```

- WORKAREA

```
          MACRO                                                    00000100
          WORKAREA                                                 00000200
          GBLB  &NOEXTRA                                           00000300
&NOEXTRA  SETB  1                                                  00000400
*         NO ADDITIONAL STACK STORAGE REQUIRED FOR THIS ROUTINE    00000500
          MEND                                                     00000600
```


5.3 Library Routine Descriptions

This section contains descriptive material for all routines in the HAL/S-FC runtime library. The routines have been grouped into seven categories. The routines within each category are described in one subsection as follows:

- 5.3.1 Arithmetic
- 5.3.2 Algebraic
- 5.3.3 Vector/Matrix
- 5.3.4 Character
- 5.3.5 Array Functions
- 5.3.6 Miscellaneous
- 5.3.7 Remote Operations

The documentation is based upon the "load module" as a basic unit. A load module is the entity created by a single invocation of the AP-101 linkage editor. It has a primary member name and may have up to 16 alias names. The primary and alias names indicate entry points to the module.

For each load module in the runtime library, an LRD form will be found in the succeeding sections. The basic LRD form is shown in Figure 1. The circled numbers in the figure are explained below.

⓪- The boxed area of the form (① - ⑦ below) contains information relating to qualities and attributes of the load module apart from any of its entry points.

①- In the upper right portion of every routine or entry point description, the name of the primary entry point will be seen. This serves as a quick reference aid in locating the documentation for a load module.

(2) - Source Member Name - The name of the member in the assembler language source PDS of the library. This name is always the same as the primary entry point name.

(3) - Size of Code Area - Each library module contains one code CSECT, regardless of the number of entry points. This number is the count of halfwords of code that would be used if the module were loaded. A module will be loaded if any one of its entry points is referenced.

(4) - Stack requirement - If a module is not an intrinsic (see (6)), it will have a requirement for runtime stack space. The minimum required will be one standard stack frame (18 Hw). The number listed on the form indicates the module's total stack requirement. If the module is an intrinsic, zero will be indicated. Individual entry points in one module cannot have different stack requirements. Therefore, the stack requirement is an attribute of the module.

(5) - Data CSECT size - If the module contains a #L CSECT, its size is indicated. Otherwise, a zero is indicated. This number shows the number of halfwords of data area that will be used if the module is loaded.

(6) - Intrinsic/Procedure - The appropriate box is marked. Entry points in a module are either all intrinsic or all procedure, hence this is a quality of the module. Sector 0 routines are noted here.

(7) - Other modules referenced - A list of other load modules referenced in EXTRN statements by this load module. If this module is loaded, the indicated modules will also be loaded.

(8) - Entry point descriptions - Following the aggregate attributes of the module in 0-7 above, the descriptions of specific entry points follow.

(9) - Primary Entry Name - The name of the code CSECT* in the module and the primary entry for the module in the library load module PDS.

* ENTRY label in the case of Sector 0 routines.

(10) - Function - A brief prose description of what this entry point does.

(11) - Invoked By - Entry points may be referenced directly from compiler-emitted code, from other library modules, or both. The appropriate boxes are marked. If the upper box is marked, an example of a HAL/S construct which results in reference to the entry point is shown. If the lower box is marked, the names of other modules which refer to this entry point are listed. If any of the other modules listed here are loaded, this module will also be brought in.

(12) - Execution Time - The time, in microseconds, needed to perform this entry point's function. These times are obtained from examinations of trace listings of simulations of the execution of the particular library routine or entry point on Version 11.3 of the GPC simulator in detailed timing mode. Times include times for referenced routines unless specifically stated.

(13) - Input Arguments - The data that the entry point receives as input is listed. "Type" indicates the nature of the data (integer, scalar, etc.). "Precision", where applicable, is generally SP for single precision and DP for double precision. "How Passed" indicates the method of communication of the data. In the case of DP scalar arguments, this field may indicate the first floating point register of an even/odd pair. "Units", when applicable, specifies the units presumed for an argument.

(14) - Output Results - The data that is considered the "answer" from the entry point. The fields are used in the same way as in (13).

(15) - Errors Detected - If invocation of this entry point can result in a Send Error SVC being executed, the error #, cause, and standard fixup for all such errors are indicated.

(16) - Comments - Any special behavior of this entry point or notes to users are entered here.

(17) - Algorithm - The steps taken by the entry point to produce its results are shown. When appropriate, references are made to other entry point descriptions for further documentation.

In addition to the basic LRD form of Figure 1, which documents module attributes and the primary entry point, an extension LRD form is used to document additional alias entry points within a module. The extension LRD is shown in Figure 2. The circled numbers are explained below:

(18) - The primary entry name of the module is displayed. This is the same name as is displayed in the basic LRD form (1) to which this extension form is appended.

(19) - Secondary Entry Name - The name of the secondary entry point being documented.

(20) - The remainder of the extension form is identical to the primary entry point description entries (10) through (17), and describe the function and interface to this entry.

(0)

(1) _____

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

(2) Source Member Name: _____ (3) Size of Code Area: _____ Hw

(4) Stack Requirement: _____ Hw (5) Data CSECT Size: _____ Hw

☐ Intrinsic (6) ☐ Procedure

(7) Other Library Modules Referenced: _____

(8) ENTRY POINT DESCRIPTIONS

(9) Primary Entry Name: _____

(10) Function:

(11) Invoked by:
☐ Compiler emitted code for HAL/S construct of the form:

☐ Other Library Modules:

(12) Execution Time (microseconds):

(13) Input Arguments:

Type	Precision	How Passed	Units

(14) Output Results:

Type	Precision	How Passed	Units

(15) Errors Detected:

Error #	Cause	Fixup

(16) Comments:

(17) Algorithm:

Figure 1: Basic LRD Form

(18) _____

(19) Secondary Entry Name: _____

(20) Function:

Invoked by:

☐ Compiler emitted code for HAL/S construct of the form:

☐ Other library modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units

Output Results:

Type	Precision	How Passed	Units

Errors Detected:

Error #	Cause	Fixup

Comments:

Algorithm:

Figure 2: Extension LRD Form

The following table shows the routines which are assigned to each group. The table contains a list of primary and secondary entry points with each secondary indented under its primary entry. With each primary entry point, basic descriptive information is shown along with the sizes of the csects in the module and the module's stack requirement. A final entry shows the timing information for the entry point. Secondary entry points have only the descriptive information and the timing for the entry. In cases where the timing information is too involved to be listed in the space available, the notice "See LRD" indicates that the detailed write-up of the module (on an LRD form in the proper subsection) should be referenced. In all cases, information in the table is taken from the LRDs and further details on the routines' performance can be found in those detailed descriptions.

ARITHMETIC ROUTINES (Section 5.3.1)

ENTRY	FUNCTION	PREC.	CODE	DATA	STACK	TIME
DMOD	MOD(D,D)	D	42	2	0	74.6
DMDVAL	MIDVAL(D,D,D)	D	20	0	18	41.4
EMOD	MOD(S,S)	S	36	2	0	46.6
IMOD	MOD(I,I)	I	20	2	0	29.4
HMOD	MOD(H,H)	H				29.4
IREM	REMAINDER(I,I)	I	14	2	0	27.0
HREM	REMAINDER(H,H)	H				27.0
ROUND	ROUND(S)	I	84	2	0	39.0
CEIL	CEILING(S)	I				See LRD
DCEIL	CEILING(D)	I				See LRD
DFLOOR	FLOOR(D)	I				See LRD
DROUND	ROUND(D)	I				33.8
DTOI	D → I	I				33.8
DTRUNC	TRUNCATE(D)	I				28.6
ETOI	S → I	I				39.0
FLOOR	FLOOR(S)	I				See LRD
TRUNC	TRUNCATE(S)	I				31.4

ALGEBRAIC ROUTINES (Section 5.3.2)

ENTRY	FUNCTION	PREC.	CODE	DATA	STACK	TIME
ACOS	ARCCOS(S)	S	102	2	24	See LRD
ASIN	ARCSIN(S)	S				See LRD
ACOSH	ARCCOSH(S)	S	36	2	20	See LRD
ASINH	ARCSINH(S)	S	64	0	20	See LRD
ATANH	ARCTANH(S)	S	58	2	18	See LRD
DACOS	ARCCOS(D)	D	150	2	18	See LRD
DASIN	ARCSIN(D)	D				See LRD
DACOSH	ARCCOSH(D)	D	42	2	22	See LRD
DASINH	ARCSINH(D)	D	94	0	22	See LRD
DATANH	ARCTANH(D)	D	90	2	18	See LRD
DATAN2	ARCTAN2(D,D)	D	194	26	18	248.4
DATAN	ARCTAN(D)	D				237.3
DEXP	EXP(D)	D	154	66	18	290.5
DLOG	LOG(D)	D	140	2	22	282.2
DPWRD	D**D	D	38	4	22	See LRD
DPWRI	D**I	D	40	2	18	See LRD
DPWRH	D**H	D				See LRD
DSIN	SIN(D)	D	102	62	20	267.0
DCOS	COS(D)	D				261.8 - 264.2
DISNH	SINH(D)	D	130	2	22	See LRD
DCOSH	COSH(D)	D				422.6
DSQRT	SQRT(D)	D	70	2	26	345.2
DTAN	TAN(D)	D	164	4	30	302.2
DTANH	TANH(D)	D	94	0	22	See LRD

ALGEBRAIC ROUTINES (CONTINUED) (Section 5.3.2)

ENTRY	FUNCTION	PREC.	CODE	DATA	STACK	TIME
EATAN2	ARCTAN2(S,S)	S	132	10	18	120.0
ATAN	ARCTAN(S)	S				116.5
EPWRE	S**S	S	32	4	22	See LRD
EPWRI	S**I	S	38	2	18	See LRD
EPWRH	S**H	S				See LRD
EXP	EXP(S)	S	108	2	18	141.8
IPWRI	I**I	I	46	2	18	See LRD
HPWRH	H**H	H				See LRD
IPWRH	I**H	I				See LRD
LOG	LOG(S)	S	90	2	18	140.5
SIN	SIN(S)	S	70	30	0	123.6 - 124.5
COS	COS(S)	S				122.1 - 123.1
SINH	SINH(S)	S	80	2	18	See LRD
COSH	COSH(S)	S				228.9
SQRT	SQRT(S)	S	48	14	0	88.3
TAN	TAN(S)	S	112	4	20	164.0
TANH	TANH(S)	S	56	0	18	See LRD

VECTOR/MATRIX ROUTINES (Section 5.3.3)

ENTRY	FUNCTION	SIZE	PREC.	CODE	DATA	STACK	TIME
MM0DNP	Scalar to Partitioned Matrix Move	n,m	D	12	0	0	6.8+n(4.0+8.0m)
MM0SNP	"	n,m	S	10	0	0	6.4+n(4.4+6.4m)
MM1DNP	Partitioned Matrix Move	n,m	D	18	0	0	10.8+n(5.4+12.2m)
MM1SNP	"	n,m	S	16	0	0	10.8+n(5.4+9.4m)
MM1TNP	"	n,m	D-S	16	0	0	10.4+n(5.8+10.6)
MM1WNP	"	n,m	S-D	18	0	0	13.6+n(5.0+11.0m)
MM6DN	Matrix Multiply	(m,n),(n,ℓ)	D	42	0	0	22.2+m(10.8+ℓ(21.2+27.n))
MM6D3	"	(3,3),(3,3)	D	32	0	0	671.6
MM6SN	"	(m,n),(n,ℓ)	S	40	0	0	22.2+m(10.8+ℓ(20.2+18.0n))
MM6S3	"	(3,3),(3,3)	S	24	0	0	409.6
MM11DN	Matrix Transpose	n,m	D	16	0	0	8.0+m(5.8+12.2n)
MM11D3	"	3,3	D	22	0	0	93.6
MM11SN	"	m,n	S	16	0	0	8.4+m(5.8+9.4n)
MM11S3	"	3,3	S	18	0	0	71.8
MM12DN	Matrix Determinant	n,n	D	150	0	22	See LRD
MM12D3	"	3,3	D	44	0	18	229.6
MM12SN	"	n,n	S	138	0	20	See LRD
MM12S3	"	3,3	S	26	0	18	116.0
MM13DN	Matrix Trace	n,n	D	10	0	0	12.0+10.2n
MM13D3	"	3,3	D	8	0	0	19.8
MM13SN	"	n,n	S	8	0	0	8.8+6.2n
MM13S3	"	3,3	S	4	0	0	9.8

VECTOR/MATRIX ROUTINES (CONTINUED) (Section 5.3.3)

ENTRY	FUNCTION	SIZE	PREC.	CODE	DATA	STACK	TIME
MM14DN	Matrix Inverse	n,n	D	258	2	20	$63.0+129.5n+43.0n^2+65.4n^3$
MM14D3	"	3,3	D	128	2	18	795.4
MM14SN	"	n,n	S	242	2	20	$52.0+39.2n+10.5n^2+54.6n^3$
MM14S3	"	3,3	S	80	2	18	458.8
MM15DN	Identity Matrix	n,n	D	18	0	0	$15.6+5.0n+11.2n^2$
MM15SN	"	n,n	S	14	0	0	$10.0+5.2n+9.6n^2$
MM17D3	Matrix to a Power	3,3	D	86	0	20	See LRD
MM17DN	"	n,n	D				See LRD
MM17S3	"	3,3	S	78	0	20	See LRD
MM17SN	"	n,n	S				See LRD
MV6DN	Matrix times Vector	(m,n),n	D	24	0	0	12.0+m(19.3+26.0n)
MV6D3	"	(3,3),3	D	22	0	0	304.4
MV6SN	"	(m,n),n	S	18	0	0	11.2+m(11.0+18.4n)
MV6S3	"	(3,3),3	S	20	0	0	137.6
VM6DN	Vector times Matrix	n,(n,m)	D	26	0	0	23.2+m(23.2+27.6n)
VM6D3	"	3,(3,3)	D	24	0	0	227.8
VM6SN	"	n,(n,m)	S	22	0	0	12.4+m(19.2+18.2n)
VM6S3	"	3,(3,3)	S	16	0	0	141.2
VO6DN	Vector Outer Product	n,m	D	20	0	0	12.8+n(5.8+24.4m)
VO6D3	"	3,3	D	22	0	0	251.0
VO6SN	"	n,m	S	20	0	0	14.2+n(5.8+14.4m)
VO6S3	"	3,3	S	20	0	0	160.6
VV0DN	Scalar to Vector Move	n	D	6	0	0	7.0+5.1n
VV0DNP	Scalar to Column Vector Move	n	D	6	0	0	7.0+7.2n

VECTOR/MATRIX ROUTINES (CONTINUED) (Section 5.3.3)

ENTRY	FUNCTION	SIZE	PREC.	CODE	DATA	STACK	TIME
VV0SN	Scalar to Vector Move	n	S	6	0	0	7.0+5.6n
VV0SNP	Scalar to Column Vector Move	n	S	6	0	0	7.0+6.0n
VV1DN	Vector Move	n	D	8	0	0	4.2+10.2n
VV1D3	"	3	D	14	0	0	25.2
VV1D3P	Column Vector Move	3	D	18	0	0	See LRD
VV1DNP	"	n	D				See LRD
VV1SN	Vector Move	n	S	8	0	0	4.2+7.8n
VV1S3	"	3	S	8	0	0	16.8
VV1S3P	Column Vector Move	3	S	14	0	0	See LRD
VV1SNP	"	n	S				See LRD
VV1TN	Vector Move	n	D-S	8	0	0	4.2+9.0n
VV1T3	"	3	D-S	12	0	0	21.2
VV1T3P	Column Vector Move	3	D-S	14	0	0	See LRD
VV1TNP	"	n	D-S				See LRD
VV1WN	Vector Move	n	S-D	10	0	0	8.4+9.0n
VV1W3	"	3	S-D	12	0	0	23.8
VV1W3P	Column Vector Move	3	S-D	18	0	0	See LRD
VV1WNP	"	n	S-D				See LRD
VV2DN	Vector Add/Matrix Add	n	D	14	0	0	8.8+20.6n
VV2D3	Vector Add	3	D	22	0	0	51.4
VV2SN	Vector Add/Matrix Add	n	S	10	0	0	8.2+13.6n
VV2S3	Vector Add	3	S	12	0	0	29.6
VV3DN	Vector Subtract/Matrix Subtract	n	D	16	0	0	6.0+22.7n

VECTOR/MATRIX ROUTINES (CONTINUED) (Section 5.3.3)

ENTRY	FUNCTION	SIZE	PREC.	CODE	DATA	STACK	TIME
VV3D3	Vector Subtract	3	D	24	0	0	55.4
VV3SN	Vector Subtract/Matrix Subtract	n	S	10	0	0	8.4+13.6n
VV3S3	Vector Subtract	3	S	12	0	0	29.6
VV4DN	Vector or Matrix Times Scalar	n	D	8	0	0	7.0+23.4n
VV4D3	Vector Times Scalar	3	D	18	0	0	68.4
VV4SN	Vector or Matrix Times Scalar	n	S	8	0	0	7.0+14.0n
VV4S3	Vector Times Scalar	3	S	12	0	0	38.4
VV5DN	Vector or Matrix Divided by Scalar	n	D	16	2	0	37.0+24.2n
VV5D3	Vector Divided by Scalar	3	D	26	2	0	98.4
VV5SN	Vector or Matrix Divided by Scalar	n	S	14	2	0	7.2+18.0n
VV5S3	Vector Divided by Scalar	3	S	18	2	0	50.6
VV6DN	Vector Dot Product	n	D	12	0	0	16.4+25.4n
VV6D3	"	3	D	16	0	0	71.8
VV6SN	"	n	S	12	0	0	15.2+16.8n
VV6S3	"	3	S	10	0	0	41.8
VV7DN	Vector or Matrix Negate	n	D	8	0	0	7.0+11.4n
VV7D3	Vector Negate	3	D	18	0	0	32.4
VV7SN	Vector or Matrix Negate	n	S	8	0	0	7.0+9.0n
VV7S3	Vector Negate	3	S	12	0	0	23.4
VV8D3	Vector Compare	3	D	12	0	0	See LRD
VV8DN	Vector or Matrix Compare	n	D				See LRD
VV8S3	Vector Compare	3	S	12	0	0	See LRD
VV8SN	Vector or Matrix Compare	n	S				See LRD

VECTOR/MATRIX ROUTINES (CONTINUED) (Section 5.3.3)

ENTRY	FUNCTION	SIZE	PREC.	CODE	DATA	STACK	TIME
VV10D3	Unit Vector	3	D	56	2	20	402.7
VV10DN	"	n	D				259.7+47.8n
VV9DN	Vector Magnitude	n	D				226.6+24.4n
VV9D3	"	3	D				300.2
VV10S3	Unit Vector	3	S	50	2	24	236.4
VV10SN	"	n	S				130.6+32.8n
VV9SN	Vector Magnitude	n	S				118.9+14.0n
VV9S3	"	3	S				168.3
VX6D3	Vector Cross Product	3	D	36	0	0	137.6
VX6S3	"	3	S	22	0	0	78.0

CHARACTER ROUTINES (Section 5.3.4)

ENTRY	FUNCTION	CODE	DATA	STACK	TIME
CASPV	Partitioned Assign to VAC	64	2	0	See LRD
CASP	Partitioned Assign				See LRD
CASV	Assign to VAC	28	0	0	29.2 (n=0) See LRD
CAS	Assign				32.0 (n=0) See LRD
CATV	Catenate into VAC	76	0	0	See LRD
CAT	Catenate into Data				See LRD
CINDEX	INDEX Function	52	0	18	See LRD
CLJSTV	LJUST	40	2	18	See LRD
CPAS	Assign to Partition	80	2	20	See LRD
CPASP	Partition Assign to Partition	16	0	146	See LRD
CPR	Compare (= or ¬=)	46	0	0	See LRD
CPRC	Compare (all relations except = and ¬=)				See LRD
CPRA	Arrayed Compare	20	0	22	See LRD
CRJSTV	RJUST	46	2	18	See LRD
CTRIMV	TRIM	94	0	18	See LRD

ARRAY ROUTINES (Section 5.3.5)

ENTRY	FUNCTION	PREC.	CODE	DATA	STACK	TIME
DMAX	MAX(DA)	D	10	0	0	See LRD
DMIN	MIN(DA)	D	10	0	0	See LRD
DPROD	PROD(DA)	D	14	0	0	See LRD
DSUM	SUM(DA)	D	6	0	0	7.2+11.6n
EMAX	MAX(SA)	S	8	0	0	See LRD
EMIN	MIN(SA)	S	8	0	0	See LRD
EPROD	PROD(SA)	S	10	0	0	See LRD
ESUM	SUM(SA)	S	6	0	0	5.2+6.6n
HMAX	MAX(HA)	H	8	0	0	See LRD
HMIN	MIN(HA)	H	8	0	0	See LRD
HPROD	PROD(HA)	H	12	0	0	See LRD
HSUM	SUM(HA)	H	6	0	0	4.4+5.4n
IMAX	MAX(IA)	I	8	0	0	See LRD
IMIN	MIN(IA)	I	8	0	0	See LRD
IPROD	PROD(IA)	I	22	0	0	See LRD
ISUM	SUM(IA)	I	6	0	0	4.4+5.4n

MISCELLANEOUS ROUTINES (Section 5.3.6)

ENTRY	FUNCTION	CODE	DATA	STACK	TIME
BTOC	Bit to Character Conversion	28	0	0	161.0 (16 bits)
CSHAPQ	Shaping Function	40	4	18	See LRD
CSLD	SUBBIT Load of Character	246	4	22	See LRD
CSLDP	SUBBIT Load of Partitioned Character				See LRD
CPSLD	Partitioned SUBBIT Load of Character				71.8
CPSLDP	Partitioned SUBBIT Load of Partitioned Character				See LRD
CSSTP	SUBBIT Store to Partitioned Character				See LRD
CPSST	Partitioned SUBBIT Store to Character				114.4
CSST	SUBBIT Store to Character				See LRD
CPSSTP	Partitioned SUBBIT Store to Partitioned Character				See LRD
CSTRUC	Structure Compare	12	0	0	5.4+10.4n
CTOB	Character to Bit Conversion	32	2	18	See LRD
CTOE	Character to SP Scalar Conversion	287	2	30	See LRD
CTOD	Character to DP Scalar Conversion				See LRD
CTOI	Character to DP Integer Conversion	104	2	20	See LRD
CTOH	Character to SP Integer Conversion				See LRD
CTOK	Character to Bit Conversion, DEC Radix				See LRD

MISCELLANEOUS ROUTINES (CONTINUED) (Section 5.3.6)

ENTRY	FUNCTION	CODE	DATA	STACK	TIME
CTOX	Character to Bit Conversion, HEX Radix	58	4	18	See LRD
CTOO	Character to Bit Conversion, OCT Radix				See LRD
DSLD	SUBBIT Load of DP Scalar	22	2	18	36.5
DSST	SUBBIT Store into DP Scalar	54	2	18	64.6
ETOC	SP Scalar to Character Conversion	278	64	20	336.9
DTOC	DP Scalar to Character Conversion				602.5
ETOH	SP Scalar to SP Integer Conversion	14	0	0	15.4
DTOH	DP Scalar to SP Integer Conversion				17.4
GTBYTE	Character Fetch	14	0	0	See LRD
ITOC	DP Integer to Character Conversion	104	0	28	254.6
HTOC	SP Integer to Character Conversion				189.6
ITOD	DP Integer to DP Scalar Conversion	20	0	0	15.6
ITOE	DP Integer to SP Scalar Conversion	6	0	0	12.0
KTOC	Bit to Character Conversion, DEC Radix	70	0	0	262.5 (16 bits)
MSTRUC	Structure Move	8	0	0	4.2+9.4n
QSHAPQ	Shaping Functions	74	0	18	42.6+31.8n
RANDOM	Random Number Generator, Uniform Dist.	46	2	18	54.4
RANDG	Random Number Generator, Gaussian Dist.				575.8
STBYTE	Character Store	22	0	0	See LRD
XTOC	Bit to Character Conversion, HEX	68	0	0	See LRD
OTOC	Bit to Character Conversion, OCT Radix				See LRD

REMOTE ROUTINES (Section 5.3.7)

	ENTRY	FUNCTION	CODE	DATA	STACK	TIME
A.	CHARACTER ROUTINES					
	CASRPV	Partitioned Assign to VAC	86	2	22	See LRD
	CASRP	Partition Assign				See LRD
	CASRV	Assign to VAC	36	0	18	See LRD
	CASR	Assign				See LRD
	CPASR	Assign to Partition	132	2	24	See LRD
	CPASRP	Partition Assign to Partition	16	0	146	See LRD
B.	STRUCTURE ROUTINES					
	CSTR	Structure Compare	18	0	18	See LRD
	MSTR	Structure Move	10	0	18	See LRD

REMOTE ROUTINES (CONTINUED) (Section 5.3.7)

	ENTRY	FUNCTION	SIZE	PREC.	CODE	DATA	STACK	TIME
C.	VECTOR AND MATRIX ROUTINES							
	MR0DNP	Scalar to Partitioned Matrix Move	n,m	D	16	0	20	22.8+n(5.6+9.8m)
	MR0SNP	"	n,m	S	16	0	20	22.8+n(5.6+8.6m)
	MR1DNP	Partitioned Matrix Move	n,m	D	22	0	20	28.4+n(8.2+15.0m)
	MR1SNP	"	n,m	S	22	0	22	28.4+n(8.2+12.6m)
	MR1TNP	"	n,m	D-S	24	0	22	31.2+n(7.6+13.8m)
	MR1WNP	"	n,m	S-D	24	0	22	32.8+n(8.2+15.8m)
	VR0DN	Scalar to Vector Move	n	D	6	0	18	16.4+9.2n
	VR0DNP	Scalar to Column Vector Move	n	D	10	0	18	21.2+10.0n
	VR0SN	Scalar to Vector Move	n	S	6	0	18	16.4+8.0n
	VR0SNP	Scalar to Column Vector Move	n	S	10	0	18	21.2+8.8n
	VR1DN	Vector Move	n	D	8	0	18	16.4+15.0n
	VR1DNP	Column Vector Move	n	D	20	0	18	See LRD
	VR1SN	Vector Move	n	S	8	0	18	16.4+12.6n
	VR1SNP	Column Vector Move	n	S	20	0	18	See LRD
	VR1TN	Vector Move	n	D-S	8	0	18	16.4+13.8n
	VR1TNP	Column Vector Move	n	D-S	20	0	18	See LRD
	VR1WN	Vector Move	n	S-D	10	0	18	20.6+13.8n
	VR1WNP	Column Vector Move	n	S-D	22	0	18	See LRD

5.3.1 Arithmetic Routine Descriptions

This subsection presents the detailed descriptions of a class of routines generally denoted as "Arithmetic". Appendix C of the HAL/S Language Specification contains a list of HAL/S functions which are implemented by the routines described here.

DMOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DMOD Size of Code Area: 42 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DMOD

Function: Calculates HAL/S MOD function in double precision.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:
MOD(A,B), where at least one of A or B is a double precision scalar.

[ ] Other Library Modules:

Execution Time (microseconds): 74.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0/F1	-
Scalar	DP	F2/F3	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0/F1	-

Errors Detected:

Error #	Cause	Fixup
19	mod domain error when B=0, A < 0	Return A

Comments: If one argument is several orders of magnitude greater than the other argument, the code sequence for A-(|B|*FLOOR(A/|B|) loses some bits of precision... it is possible to have the result of A(mod B) be greater than |B|.

Algorithm:
Use |B|. If B=0, then check A for a possible mod domain error. If A ≥ 0, then return A. If A < 0, signal error and return negative A. If not equal, then use MOD(A,B) = A-(|B|*FLOOR(A/|B|)). First, get X = A/|B|. If X = 0, then exit with result 0 (0 mod(B) = 0). The FLOOR(X) is different for negative and positive X. If X < 0, round X down past next smaller negative integer by subtracting .9999999999999999. Then subtract BIGNUM(X'4E80000000000000') to get rid of the decimal places and leave only an integer value. Add BIGNUM to normalize the integer value. If X > 0, no rounding is done; BIGNUM is first added, then subtracted.

For arguments that are orders of magnitude apart, it is necessary to check that the result is ≤ |B|. If RESULT > |B|, return |B|.

DMOD

Comments (Cont'd.)

Registers Unsafe Across Call: R4,F0,F1,F2,F3,F4,F5.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DMDVAL Size of Code Area: 20 Hw

Stack Requirement: 13 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DMDVAL

Function: Finds mid value of three double precision scalar arguments.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

MIDVAL(A,B,C) where A,B,C are double precision scalars.

☐ Other Library Modules:

Execution Time (microseconds): 41.4

Input Arguments:

Type	Precision	How Passed	Units
scalar	DP	F0	
scalar	DP	F2	
scalar	DP	F4	

Output Results:

Type	Precision	How Passed	Units
scalar	DP	F0	

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

```
IF A = B THEN RETURN A;
IF A < B THEN DO;
   IF B <= C THEN RETURN B;
   ELSE IF A <= C THEN RETURN C;
     ELSE RETURN A;
END;
ELSE DO;
   IF C <= B THEN RETURN B;
   ELSE IF C <= A THEN RETURN C;
     ELSE RETURN A;
END;
```

EMOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EMOD Size of Code Area: 36 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EMOD

Function: Calculates HAL/S MOD function in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
MOD(A,B), where A and B are single precision scalars.

☐ Other Library Modules:

Execution Time (microseconds): 46.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Scalar	SP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
19	mod domain error when B=0,A<0.	Return A

Comments: See DMOD.
Registers Unsafe Across Call: R4,F0,F1,F2,F3,F4,F5.

Algorithm: See DMOD.

IMOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IMOD Size of Code Area: 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IMOD

Function: Calculates A Mod(B). For $A \geq 0$, the mod can be defined as $MOD=|B|-\lfloor A-(n+1)|B| \rfloor$ where n is an integer and $\lfloor A-(n+1)|B| \rfloor < 0 \leq \lfloor A-n|B| \rfloor$. For $A < 0$, $MOD=|B|-|A+(n-1)|B||$, where n is an integer and $\lfloor A+(n-1)|B| \rfloor \leq 0 < \lfloor A+n|B| \rfloor$.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
MOD(A,B), A and B are both integers and at least A or B is a fullword integer value.

☐ Other Library Modules:

Execution Time (microseconds): 29.4

Input Arguments:

Type	Precision	How Passed	Units
Integer	DP	R5	-
Integer	DP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
19	MOD not defined for first arg < 0 and second arg = 0.	Return first arg.

Comments:
Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:
Use $|B|$. If B=0, then check for possible error; if $A \geq 0$, then result is A. If $A < 0$, error. For $B \neq 0$, the mod is found using one of two formulae, depending on the value of A. For $A \geq 0$, $MOD(A,B) = A-[|B|*(A/|B|)]$. For $A < 0$, $MOD(A,B) = A-|B|*(A/|B|)+|B|$. For all values of A, the result is always non-negative.
For $A \geq 0$, MOD = REMAINDER(A,B). These equations are used because AP-101 division (scalar or integer) does not yield a remainder.

Secondary Entry Name: HMOD

Function: Performs HAL/S MOD(A,B) where both A and B are single precision integers.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
MOD(A,B), where A and B are both single precision integers.

[ ] Other library modules:

Execution Time (microseconds): 29.4

Input Arguments:

Type	Precision	How Passed	Units
Integer	SP	R5	-
Integer	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
Same as IMOD		

Comments:
Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:
Same as IMOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IREM Size of Code Area: 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced:

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IREM

Function: Calculates integer remainder of (A,B).

Invoked by:
[X] Compiler emitted code for HAL/S construct of the form:
REMAINDER(A/B), where A and B are both integers and at least one of A or B is double precision.
[ ] Other Library Modules:

Execution Time (microseconds): 27.0

Input Arguments:

Type	Precision	How Passed	Units
Integer	DP } one can be SP	R5	-
Integer	DP } one can be SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
16	zero denominator (B)	Return A

Comments:
Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:
If B=0, then error. For B ≠ 0, the remainder is found using REMAINDER(A,B) =[A- B*(A/B)]. The result can be negative.

IREM

Secondary Entry Name: HREM

Function:
Calculates integer remainder of A/B.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
REMAINDER(A,B), where A and B are both single precision integers.

[ ] Other library modules:

Execution Time (microseconds): 27.0

Input Arguments:

Type	Precision	How Passed	Units
Integer	SP	R5	-
Integer	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
16	zero denominator (B)	Return A

Comments:
Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:

Same as IREM.

ROUND

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ROUND Size of Code Area: 84 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic SECTOR 0 [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ROUND

Function: Converts single precision scalar to fullword integer.

Invoked by:
[X] Compiler emitted code for HAL/S construct of the form:
ROUND(X) where X is a single precision scalar.

[X] Other Library Modules: QSHAPQ

Execution Time (microseconds): 39.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Set to max/min representable value: Negmax = X'80000000' Posmax = X'7FFFFFFF'

Comments:

See DROUND. Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Second register of a floating point register pair is cleared then routine merges into the double precision float-to-fix routine, DROUND.

Secondary Entry Name: CEIL

Function: Performs HAL/S CEILING function: Returns smallest integer ≥ the argument.

Invoked by:

☑ Compiler emitted code for HAL/S construct of the form: CEILING(X), where X is a single precision scalar.

☐ Other library modules:

Execution Time (microseconds): 31.4 if X ≥ 0
40.8 if X < 0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

See DCEIL. Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Second register of floating point register pair is cleared, then routine merges with DCEIL.

Secondary Entry Name: DCEIL

Function: Performs HAL/S CEILING function: Finds the smallest integer ≥ the argument.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: CEILING(X), where X is a double precision scalar.

☐ Other library modules:

Execution Time (microseconds): 26.6 if X ≥ 0
36.0 if X < 0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Output Results:

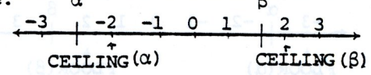
Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments: Negative args become less negative after CEILING; positive args more positive.

Algorithm:



(Cont'd. at bottom of page)

Same as DROUND, except positive arguments are rounded up by almost 1. Negative arguments are not rounded.

Comments: (Continued)

Registers Unsafe Across Call: R4,R5,F0,F1.

Secondary Entry Name: DFLOOR

Function: Performs HAL/S FLOOR function: Finds the largest integer $\leq$ the argument.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: FLOOR(X), where X is a double precision scalar.

☐ Other library modules:

Execution Time (microseconds): 27.0 if X $\geq$ 0
36.4 if X $<$ 0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Output Results:

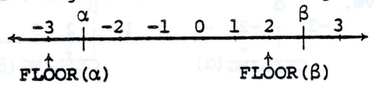
Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments: Negative arguments become more negative, positive arguments less positive. (Cont'd. at bottom of page)

Algorithm:



Same as DROUND, except argument is rounded down by almost 1 (X'40FFFFFFFFFFFFFF') if negative. Positive arguments are not rounded.

Comments: (Cont'd.)

Registers Unsafe Across Call: R4,R5,F0,F1.

Secondary Entry Name: DROUND

Function: Converts double precision scalar to fullword integer.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: ROUND(X), where X is a double precision scalar.

☐ Other library modules:

Execution Time (microseconds): 33.8

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

Negative arguments are converted to the next more negative integer value; positive args to the next greater positive integer value, unless the (Con't on bottom of page).

Algorithm:

The argument is checked for negative/not negative. If the argument is negative, the value is rounded down by subtracting just under ½. The resulting value is then checked against MAXNEG(X'C880000000FFFFFF'). If within the legal range, the integer part of the scalar is shifted to the second register of the floating point register pair. This remaining integer value is then put in a fixed point register and complemented to leave it in the correct two's-complement fixed point form. If the argument is not negative, the value is rounded up by adding almost ½, and the resulting value is compared to MAXPOS (X'487FFFFFFFFFFFFF'). Then, as with negative values, it is shifted to leave the integer part in floating point format and loaded into a fixed point register.

Comments (Con't): original argument is an integer (argument rounded up or down by not quite 1 before truncating decimal places).

Registers Unsafe Across Call: R4,R5,F0,F1.

Secondary Entry Name: DTOI

Function: Converts double precision scalar to fullword integer.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
I = D; where I is a double precision integer, and
D is a double precision scalar.

[ ] Other library modules:

Execution Time (microseconds): 33.8

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Return either: POSMAX or NEGMAX

Comments: DTOI is identical entry point to DROUND.
Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm: Same as DROUND.

Secondary Entry Name: DTRUNC

Function: Performs HAL/S TRUNCATE function: Finds the signed value that is the largest integer ≤ absolute value of the argument.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
TRUNCATE(X), where X is a double precision scalar.

[ ] Other library modules:

Execution Time (microseconds): 28.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

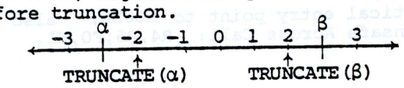
Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Return either: Posmax: X'7FFFFFFF' or Negmax: X'80000000'

Comments: After truncation, negative and positive arguments are closer to 0; no rounding done before truncation.

(Continued at bottom of page)

Algorithm:



Same as DROUND, except argument is not rounded up or down.

Comments: (Continued)

Registers Unsafe Across Call: R4,R5,F0,F1.

Secondary Entry Name: ETOI

Function:
Converts single precision scalar to fullword integer.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
I=S where I is a double precision integer, S is a single precision scalar.

☐ Other library modules:

Execution Time (microseconds): 39.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Return either: Posmax = X'7FFFFFFF' Negmax = X'80000000'

Comments:
ETOI is identical entry point to ROUND; also see DTOI
Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Same as ROUND.

Secondary Entry Name: FLOOR

Function: Performs HAL/S FLOOR function: Returns largest integer ≤ the argument.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
FLOOR(X), where X is a single precision scalar.

☐ Other library modules:

Execution Time (microseconds): 31.2 if X ≥ 0
40.6 if X < 0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Returns either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:
See DFLOOR
Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Second register of floating point register pair is cleared, then routine merges with DFLOOR.

Secondary Entry Name: TRUNC

Function: Performs HAL/S TRUNCATE function: Returns signed value that is the largest integer ≤ absolute value of the argument.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
TRUNCATE(X) where X is a single precision scalar.

☐ Other library modules:

Execution Time (microseconds): 31.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

See DTRUNC
Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Second register of floating point register pair is cleared, then routine merges with DTRUNC.

5.3.2 Algebraic Routine Description

This subsection presents the detailed descriptions of "Algebraic" routines as defined in Appendix C of the HAL/S Language Specification.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ACOS Size of Code Area: 102 Hw

Stack Requirement: 24 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: SQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ACOS

Function: Computes arc-cosine(x) of scalar argument.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ARCCOS(X), where X is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): $.5 < |X| \leq 1$: 225.5
$2.441406252 \times 10^{-4} < |X| \leq .5$: 132.7
$|X| \leq 2.441406252 \times 10^{-4}$: 71.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	radians

Errors Detected:

Error #	Cause	Fixup
10	argument outside range -1 to 1.	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: ACOS(X) is computed as $\pi/2$ - ARCSIN(X).

Secondary Entry Name: ASIN

Function:
Computes arc-sine of scalar argument.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ARCSIN(X), where X is a single precision scalar.

☐ Other library modules:

Execution Time (microseconds): $.5 < |X| \leq 1$: 227.6
$2.441406252 \times 10^{-4} < |X| \leq .5$: 118.4
$|X| \leq 2.441406252 \times 10^{-4}$: 57.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	radians

Errors Detected:

Error #	Cause	Fixup
10	argument outside legal range.	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: The value of X is restricted to $0 \leq X \leq 1$ by using the identity $\arcsin(-X) = -\arcsin(X)$, and further to $0 \leq X \leq .5$ by the identity $\arcsin(X) = \pi/2 - 2\arcsin(\sqrt{\frac{1-X}{2}})$. In this range, arcsin(X) is computed as a truncated continued fraction in X^2, multiplied by X. The form of the approximation is:

$$\arcsin(X) \approx X + \cfrac{d_1 x^3}{c_1 + x^2 + \cfrac{d_2}{c_2 + x^2}}$$

where the values of the constants are:

c_1 = X'C13B446A' = -3.7042025
c_2 = X'C11DB034' = -1.8555182
d_1 = X'C08143C7' = -0.5049404
d_2 = X'C11406BF' = -1.2516474

ACOSH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ACOSH Size of Code Area: 36 Hw

Stack Requirement: 20 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: LOG, SQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ACOSH

Function:

Computes hyperbolic arc-cosine in single precsion.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

ARCCOSH(x), where x is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 1.6777722E+7 ≤ X: 124.2
1 ≤ X < 1.6777722E+7: 297.3

Input Arguments:

Type	Precision	How Passed	Units
scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
59	ARG < 1	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external SQRT and LOG functions:

$$\text{arccosh}(x) = \log(x + \sqrt{x^2-1})$$

ASINH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ASINH Size of Code Area: 64 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: SQRT, LOG

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ASINH

Function:

Computes hyperbolic arc-sine in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

ARCSINH(X), where X is a scalar.

☐ Other Library Modules:

Execution Time (microseconds): (See below)

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
none		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external SQRT and LOG routines:

$$\text{arcsinh}(X) = \log(X + \sqrt{X^2 + 1})$$

Execution Time: |X| < 8.8721751E-4: 31.5
|X| ≥ 1.6777722E7: 141.2
8.8721751E-4 ≤ |X| < 2.1632855E-1: 85.4
2.163255E-1 < |X| < 1.6777722E+7: 314.1

ATANH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ATANH Size of Code Area: 58 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: LOG

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ATANH

Function:

Computes hyperbolic arc-tangent in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ARCTANH(X), where X is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): $|X| < 4.113892E\text{-}5$: 33.9
$4.113892E\text{-}5 \leq |X| \leq 1.875E\text{-}1$: 85.7
$1.875E\text{-}1 < |X|$: 228.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
60	argument outside range: $-1 < X < 1$	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external LOG function,

$$\text{arctanh}(S) = \tfrac{1}{2} \log((1 + X)/(1 - X)).$$

Error #60 is sent if $1 - X = 0$, or if $(1 + X)/(1 - X) \leq 0$.
which, taken together, are equivalent to the requirement $-1 < X < 1$.

DACOS

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DACOS Size of Code Area: 150 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: DSQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DACOS

Function: Computes ARCCOS(X) in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ARCCOS(X), where X is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): $|X| \leq 3.7252907E\text{-}7$: 89.1
$3.7252907E\text{-}7 < |X| \leq .5$: 263.1
$.5 < |X| \leq 1$: 460.5 (79.7 in odd cases)

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0 -	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	radians

Errors Detected:

Error #	Cause	Fixup
10	argument outside range $-1 \leq X \leq 1$	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Computed as $\pi/2$ - ARCSIN(X)

Secondary Entry Name: DASIN

Function:

Computes ARCSIN(X) in double precision.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
ARCSIN(X), where X is a double precision scalar.

[ ] Other library modules:

Execution Time (microseconds): $|X| \leq 3.7252907E\text{-}7$: 64.1
$3.7252907E\text{-}7 < |X| \leq .5$: 238.1
$.5 < |X| \leq 1$: 470.3 (89.5 in odd cases)

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	radians

Errors Detected:

Error #	Cause	Fixup
10	argument outside range $-1 \leq X \leq 1$	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: The value of X is restricted to $0 \leq X \leq 1$ by using the identity

$$\arcsin(-X) = \arcsin(X),$$

and further to $0 \leq X < \frac{1}{2}$ by using the identity

$$\arcsin(X) = \pi/2 - 2\arcsin\sqrt{\frac{1-X}{2}}.$$

Within this range, arcsin(X) is computed as a truncated continued fraction in X^2, multiplied by X.

The form of the approximation is:

$$\arcsin(X) = X + X^3\left(C_1 + \cfrac{d_1}{X^2 + C_2 + \cfrac{d_2}{X^2 + C_3 + \cfrac{d_3}{X^2 + C_4 + \cfrac{d_4}{X^2 + C_5}}}}\right)$$

(Continued on next page)

DACOS

Algorithm (Con't)

where the values of the constants are:

C_1 = X'3F180CD96B42A610' = .00587162904063511

d_1 = X'C07FE6DD798CBF27' =-.49961647241138661

C_2 = X'C1470EC5E7C7075C' =-4.44110670602864049

d_2 = X'C1489A752C6A6B54' =-4.53770940160639666

C_3 = X'C13A5496A02A788D' =-3.64565146031194167

d_3 = X'C06B411D9ED01722' =-.41896233680025977

C_4 = X'C11BFB2E6EB617AA' =-1.74882357832528117

d_4 = X'BF99119272C87E78' =-.03737027365107758

C_5 = X'C11323D9C96F1661' =-1.19625261960154476

DACOSH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DACOSH Size of Code Area: 42 Hw

Stack Requirement: 22 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: DLOG, DSQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DACOSH

Function: Computes hyperbolic arc-cosine in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

ARCCOSH(X), where X is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): $1 \leq X < 6.7108869E+7$: 403.4
$6.7108869E+7 < X$: 332.4

Input Arguments:

Type	Precision	How Passed	Units
scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
59	argument < 1	return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external DSQRT and DLOG functions:

$$\text{arccosh}(x) = \log(x + \sqrt{x^2-1})$$

DASINH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DASINH Size of Code Area: 94 Hw

Stack Requirement: 22 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: DSQRT, DLOG

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DASINH

Function:

Computes hyperbolic arc-sine in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ARCSINH(X), where X is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): (See below).

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external DSQRT and DLOG function:

$$\text{arcsinh}(X) = \log(X + \sqrt{X^2 + 1})$$

Execution Time:
$|X| \leq 1.353860E-8$: 33.6
$6.7108864E+7 \leq |X|$: 348.2
$1.353860E-8 < |X| \leq 6.25E-2$: 185.4
$6.25E-02 < |X| < 6.7108864E+7$: 570.8

DATANH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DATANH Size of Code Area: 90 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: DLOG

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DATANH

Function:

Computes hyperbolic arc-tangent in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ARCTANH(X), where X is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): $|X| < 1.0774559E-8$: 42.6
$1.0774559E-8 \leq |X| < 6.25E-2$: 186.6
$6.25E-2 \leq |X|$: 399.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
60	Argument outside range: $-1 < X < 1$	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external DLOG library function,

$$\text{arctanh}(X) = \tfrac{1}{2} \log ((1 + X)/(1-X)).$$

DATAN2

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DATAN2 Size of Code Area: 194 Hw

Stack Requirement: 18 Hw Data CSECT Size: 26 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DATAN2

Function: Computes arctan by fraction approximation in the range $(-\pi, \pi)$ in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ARCTAN2(X,Y), where X and Y are double precision scalar corresponding to sine and cosine respectively of the intended arc tangent argument.

☐ Other Library Modules:

Execution Time (microseconds): 248.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar (sin)	DP	F0	-
Scalar (cos)	DP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
62	arg 1 = arg 2 = 0	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Same algorithm as EATAN2, but values of constants and the fractional approximation formula is different for the double precision, as follows.

Again, $Z = \frac{|\sin x|}{|\cos x|}$. Special cases - (1) If $\cos x < 0$ and $Z < (16)^{-14}$, return $\pm\pi$. (2) $\sin x = \cos x = 0$, signal error and return 0. (3) $\sin x \neq 0$, $\cos x = 0$, return $\pm \pi/2$. (4) $\sin x \neq 0$, $\cos x \neq 0$, but $Z > (16)^{14}$, return $\pm \pi/2$. (5) If $Z < (16)^{-7}$, return Z.

(Continued on next page)

DATAN2

Algorithm (Con't)

The fractional approximation after reduction of Z to $\leq$ tan 15° is:

$$\text{Tan}^{-1}(Z) = Z + Z \cdot Z^2 \cdot F, \text{ where}$$

$$F = C1 + C2/(Z^2 + C3 + C4/[Z^2 + C5 + (C6/(Z^2 + C7))]).$$

C1 = X'BF1E31FF1784B965' (-0.7371899082768562E-2)

C2 = X'C0ACDB34C0D1B35D' (-0.6752198191404210)

C3 = X'412B7CE45AF5C165' (0.2717991214096480E+1)

C4 = X'C11A8F923B178C78' (-0.1660051565960002E+1)

C5 = X'412AB4FD5D433FF6' (0.2669186939532663E+1)

C6 = X'C02298BB68CFD869' (-0.1351430064094942)

C7 = X'41154CEE8B70CA99' (0.1331282181443987E+1)

As in EATAN2, the intermediate result is adjusted to the proper section in the first quadrant, as follows:

(original) Z $\leq$ tan 15° → + 0

tan 15° < Z $\leq$ 1 → + π/6

1/Z $\leq$ tan 15° → (-π/2 + 1) then -1 (to preserve signif. bits)

tan 15° < 1/Z $\leq$ 1 → (-π/3 + 1) then -1 (to preserve signif. bits)

The resulting angle is adjusted to the proper quadrant as in EATAN2 (according to sign of sin x and cos x).

Secondary Entry Name: DATAN

Function: Computes arc tangent by fractional approximation in the range (-π/2, +π/2) in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: ARCTAN(X), where X is a double precision scalar.

☐ Other library modules:

Execution Time (microseconds): 237.3

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Same as ARCTAN, but see DATAN2 for changes in values of DP constants and TAN^{-1} formula.

DEXP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DEXP Size of Code Area: 154 Hw

Stack Requirement: 18 Hw Data CSECT Size: 66 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DEXP

Function:
Computes e^X in double precision.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:

EXP(X), where X is a double precision scalar.

☒ Other Library Modules:

DPWRD, DSINH, DTANH

Execution Time (microseconds): 290.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
6	Argument outside range $X \leq 174.6731$	Return maximum positive floating point number

Comments:

Gives underflow if argument to small - no error number.
Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm: First, decompose X as P'log2 + R', where P' is the integer part and first hexadecimal place of the result of dividing the high-order part of X by LOG2H, which is a single precision approximation to log2, rounded up. This is done in 80-bit precision in order to yield a true 56-bit value for R', by expressing log2 = LOG2H + LOG2L, where LOG2L is a double precision scalar. R' has the same sign as X, and |R'| might be slightly > $\frac{log2}{16}$.

(Continued on next page)

DEXP

Algorithm (Con't)

Now, if R' > 0, subtract $\frac{log2}{16}$ from it until it becomes ≤ 0, each time adding $\frac{1}{16}$ to P'. If R' $\leq -\frac{log2}{16}$, add $\frac{log2}{16}$ to it until it becomes $> -\frac{log2}{16}$, each time subtracting $\frac{1}{16}$ from P'.

At the end of this, we have

$$X = P \cdot \log2 + R, \text{ P an integral multiple of } 1/16, \text{ and } -\frac{log2}{16} < R \leq 0.$$

Represent P as $4A - B - \frac{C}{16}$, where A, B, and C are integers, $0 \leq B \leq 3$, $0 \leq C \leq 15$. Then:

$$e^X = 16^A \cdot 2^{-B} \cdot 2^{-\frac{C}{16}} \cdot e^R$$

To calculate this, we compute e^R with a polynomial approximation of the form:

$$e^r = 1 + c_1 r + c_2 r^2 + c_3 r^3 + c_r r^4 + c_5 r^5 + c_6 r^6$$

where the values of the constants are:

c_1 = X'40FFFFFFFFFFFCFC' = .9999999999999892

c_2 = X'407FFFFFFFAB64A' = .4999999999951906

c_3 = X'402AAAAAA794AA99' = .1666666659481656

c_4 = X'3FAAAA9D6AC1D734' = .0416666173078875

c_5 = X'3F2220559A15E158' = .00833161772003906

c_6 = X'3E591893' = .001359497

Then, $2^{-\frac{C}{16}}$ is computed by table lookup, 2^{-B} by shifting, and 16^A by adding A to the exponent of the answer.

DLOG

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DLOG Size of Code Area: 140 Hw

Stack Requirement: 22 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DLOG

Function:

Computes log(X) in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
LOG(X), where X is a double precision scalar.

☒ Other Library Modules:
DPWRD, DASINH, DATANH, DACOSH

Execution Time (microseconds): 282.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
7	argument outside range X > 0.	If X < 0 return log(|X|); if X=0, return maximum negative floating point number.

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: We write $X = 16^P \cdot 2^{-Q} \cdot M$, where $\frac{1}{2} \le M < 1$, P, Q are integers, $0 \le Q \le 3$. P, Q, and M are found by fixed-point calculations. Define

$A = 1$, $B = 0$, if $M > \frac{1}{2}\sqrt{2}/2$, and $A = \frac{1}{2}$, $B = 1$ otherwise. Let $Z = (M-A)/(M+A)$. Then

$$\log(X) = (4P-Q-B)\log(2)+\log((1+Z)/(1-Z))$$

(Continued on next page)

DLOG

Algorithm (Con't)

log((1+Z)/(1-Z)) is computed by an approximation of the form:

$$W + C_1 W^3 \left(W^2 + C_2 + \frac{C_3}{W^2 + C_4 + \frac{C_5}{W^2 + C_6}} \right)$$

where W = 2Z, and the values of the constants are:

C_1 = X'3DDABB6C9F18C6DD' = 0.2085992109128247E-3

C_2 = X'422FC604E13C20FE' = 0.4777351196020117E+2

C_3 = X'C38E5A1C55CEB1C4' = -0.2287631917769813E+4

C_4 = X'C16F2A64DDFCC1FD' = -6.947850100648906

C_5 = X'C12A017578F548D1' = -2.625356171124214

C_6 = X'C158FA4E0E40C0A5' = -5.561109595943017

DPWRD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DPWRD Size of Code Area: 38 Hw

Stack Requirement: 22 Hw Data CSECT Size: 4 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: DLOG, DEXP, DSQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DPWRD

Function:
Performs exponentiation of double precision scalar to double precision power.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
X**Y, where X and Y are scalars and at least X or Y is double precision.

☐ Other Library Modules:

Execution Time (microseconds): If Y = .5: 238.4
If Y ≠ .5: 635.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	DP	F0	-
Scalar (exponent)	DP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	base=0; exponent ≤ 0	Return 0.0
24	base < 0	use |base|

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: If exponent = 0.5, compute $X^{0.5}$ as $\sqrt{X}$, otherwise

$X^Y = e^{Y \log X}$, using the external DEXP and DLOG functions.

DPWRI

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DPWRI Size of Code Area: 40 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DPWRI

Function:
Exponentiation of a double precision scalar to a fullword integer power.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
X**I, where X is a double precision scalar; I is a double precision integer.

☐ Other Library Modules:

Execution Time (microseconds): (See next page)

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	DP	F0	-
Integer (exponent)	DP	R6	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power ≤ 0	Return 0.0

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm: If I is the fullword exponent, D the base, write

$$I = \Sigma_{i=0}^{32} e_i 2^i, \quad \text{where } e_i = 0 \text{ or } 1.$$

Then:

$$D^I = D^{\Sigma_i e_i 2^i} = \pi_{i=0}^{32} D^{e_i 2^i} = \pi_{e_i=1} D^{2^i}, \text{ if any } e_i = 1, \text{ and } = 1 \text{ otherwise.}$$

(Continued on next page)

DPWRI

Execution Time (microseconds):

If exponent $\geq$ 0: 40.8 + (n-1) 23.2 + 13.0m

If exponent $\leq$ 0: 64.0 + (n-1) 23.2 + 13.0m

where m = number of 1's in binary representation of |exponent|.

n = number of significant digits in binary representation of |exponent|.

Algorithm (Con't)

To compute $\pi_{e_i=1} D^{2^i}$, it is only necessary to compute successively

$D^{2^i} = D, D^2, D^4, D^8, \ldots$, and multiply the result by D^{2^i} whenever the i-th bit of the exponent is 1. This is determined by shifting bits one by one out of the exponent, and testing each one for a value of one. The loop terminates when the remaining part of the exponent is zero.

Operations are done on absolute value of exponent. If exponent was negative, the reciprocal of the result is taken as the final result.

Secondary Entry Name: DPWRH

Function:
Exponentiation of a double precision scalar to a halfword integer power.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
X**I, where X is a double precision scalar; I is a single precision integer.

☐ Other library modules:

Execution Time (microseconds): Same as DPWRI except constants are
exponent $\geq$ 0: 41.4
exponent $<$ 0: 64.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	DP	F0	-
Integer (exponent)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power $\leq$ 0	Return 0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm: The halfword exponent is shifted right to convert it to a fullword, then the DPWRI algorithm is used.

DSIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSIN Size of Code Area: 102 Hw

Stack Requirement: 20 Hw Data CSECT Size: 62 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSIN

Function:

Computes sine(X) in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: SIN(X), where X is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 267.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	radians

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
8	Argument outside range: $\|X\| < \pi \cdot 2^{50}$	Return $\frac{\sqrt{2}}{2}$

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Let $|X| \cdot \frac{4}{\pi} = Q+R$, where Q is an integer and R a fraction.

Add 4 to Q if the sine is desired and X < 0, and add 2 to Q if the cosine is desired.

Since $\sin(-x) = \sin(+x)$, and $\cos(x) = \sin(\frac{\pi}{2}+x)$, this reduces the problem to computing $\sin(x)$ for $X \geq 0$.

(Continued on next page)

DSIN

Algorithm (Con't)

Since Q has been adjusted, it is only necessary to compute $\sin(\frac{\pi}{4}(Q+R))$. If Q'=Q mod 8, then this is equal to $\sin(\frac{\pi}{4}(Q'+R))$. The eight cases of this yield, through simple trigonometric identities:

Q'=0: $\sin(x) = \sin(R \cdot \frac{\pi}{4})$

Q'=1: $\sin(x) = \cos((1-R) \cdot \frac{\pi}{4})$

Q'=2: $\sin(x) = \cos(R \cdot \frac{\pi}{4})$

Q'=3: $\sin(x) = \sin((1-R) \cdot \frac{\pi}{4})$

Q'=4: $\sin(x) = -\sin(R \cdot \frac{\pi}{4})$

Q'=5: $\sin(x) = -\cos((1-R) \cdot \frac{\pi}{4})$

Q'=6: $\sin(x) = -\cos(R \cdot \frac{\pi}{4})$

Q'=7: $\sin(x) = -\sin((1-R) \cdot \frac{\pi}{4})$

Thus, if we let R'=R in octants 0, 2, 4, 6, and R'=1-R in octants 1, 3, 5, 7. We need only compute

$$\sin(R' \cdot \frac{\pi}{4})$$

in octants 0, 3, 4, 7, and $\cos(R' \cdot \frac{\pi}{4})$ in 1, 2, 5, 6, and take the negative value in octants 4, 5, 6, 7.

$\sin(R' \cdot \frac{\pi}{4})$ and $\cos(R' \cdot \frac{\pi}{4})$ are computed by polynomial approximations.

The form of the polynomial approximation for sine is:

$$\sin(R' \cdot \tfrac{\pi}{4}) = R'(C_0 + C_1 R'^2 + C_2 R'^4 + C_3 R'^6 + C_4 R'^8 + C_5 R'^{10} + C_6 R'^{12})$$

where the values of the constants are:

C_0 = X'40C90FDAA22168C2' = .78539816339744831

C_1 = X'C014ABBCE625BE41' = -.080745512188280536

C_2 = X'3EA335E33BAC3FBD' = 2.4903945701888438E-3

C_3 = X'BD265A599C5CB632' = -3.65762041589138 72E-5

DSIN

Algorithm (Con't)

C_4 = X'3B541E0BF684B527' = 3.1336162254333759E-7

C_5 = X'B978C01C6BEF8CB3' = -1.7571500746935669E-9

C_6 = X'3778FCE0E5AD1685' = 6.8773605709403589E-12

The form of the polynomial approximation for cosine is:

$$\cos(R' \cdot \tfrac{\pi}{4}) = 1 + C_1 R'^2 + C_2 \cdot R'^4 + C_3 \cdot R'^6 + C_4 \cdot R'^8 + C_5 \cdot R'^{10} + C_6 \cdot R'^{12} + C_7 \cdot R'^{14}$$

where the values of the constants are:

C_1 = X'C04EF4F326F91777' = -.30842513753404242

C_2 = X'3F40F07C20606AB1' = 1.5854344243815420E-2

C_3 = X'BE155D3C7E3C90F8' = -3.2599188692673765E-4

C_4 = X'3C3C3EA0D06ABC29' = 3.5908604460279520E-6

C_5 = X'BA69B47B1E41AEF6' = -2.4611364033652271E-8

C_6 = X'387E731045017594' = 1.1500512028186245E-10

C_7 = X'B66C992E84B6AA37' = -3.8581890061323055E-13

Secondary Entry Name: DCOS

Function:
Computes cosine(x) in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
COS(X), where X is a double precision scalar.

☐ Other library modules:

Execution Time (microseconds): $-\pi \leq x \leq \pi$: 261.8

$X > \pi$ or $x < -\pi$: 264.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	radians

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
8	Argument outside range $\lvert X \rvert < \pi.2^{50}$	Return $\frac{\sqrt{2}}{2}$

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

See DSIN algorithm.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSINH Size of Code Area: 130 Hw

Stack Requirement: 22 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☑ Procedure

Other Library Modules Referenced: DEXP

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSINH

Function:
Computes hyperbolic sine in double precision.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:

SINH(X), where X is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds):
8.81374E-1 ≤ |X| < 1.75366E+2: 434.1
2.063017E-10 ≤ |X| < 8.81374E-01: 196.7
|X| < 2.063017E-10: 45.8

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
9	Argument outside range |X| ≤ 175.366	Return maximum positive floating point number.

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: If |X| < 1.626459E-10, then sinh(x) = x. If 1.626569E-10 ≤ |X| < .881374. Then sinh(x) is computed via a polynomial approximation.

The form of the polynomial is:

$$\sinh(x) = X + C_1X^3 + C_2X^5 + C_3X^7 + C_4X^9 + C_5X^{11} + C_6X^{13}$$

where the values of the constants are:

(Continued on next page)

DSINH

Algorithm (con't)

C_1 = X'402AAAAAAAAAA4D' = 0.1666666666666653

C_2 = X'3F2222222222BACE' = 0.8333333333367232E-2

C_3 = X'3DD00D00CB06A6F5' = 1.984126981270711E-4

C_4 = X'3C2E3BC881345D91' = 2.755733025610683E-6

C_5 = X'3A6B96B8975A1636' = 2.504995887597646E-8

C_6 = X'38B2D4C184418A97' = 1.626459177981471E-10

Otherwise, sinh(|x|) or cosh(|x|) is calculated using EXP. The number V, equal to 0.4995050, is introduced to control rounding errors and the formula is as follows:

$$\sinh(x) = \frac{1}{2v}\left(e^{(x+\log v)} - \frac{v^2}{e^{(x+\log v)}}\right)$$

$$\cosh(x) = \frac{1}{2v}\left(e^{(x+\log v)} + \frac{v^2}{e^{(x+\log v)}}\right)$$

The identities sinh(-x) = -sinh(x) and cosh(-x) = cosh(x) are used to recover sinh(x) and cosh(x) from sinh(|x|) and cosh(|x|).

Secondary Entry Name: DCOSH

Function:

Computes hyperbolic cosine in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
COSH(X), where X is a double precision scalar.

☐ Other library modules:

Execution Time (microseconds): $|X| \leq 1.75366E+2$: 422.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
9	Argument outside range $\|X\| \leq 175.366$	Return maximum positive floating point number

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

See DSINH algorithm.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSQRT Size of Code Area: 70 Hw

Stack Requirement: 26 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSQRT

Function:

Computes square root in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
SQRT(X), where X is a double precision scalar.

☒ Other Library Modules:
DACOS, DASINH, DPWRD, VV10D3, DACOSH

Execution Time (microseconds): 345.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
5	Argument outside of range $X \geq 0$	Return sqrt(|X|)

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Let $X = 16^{2P+Q} \cdot M$, where P, Q are integers, Q = 0 or 1, and $\frac{1}{16} \leq M < 1$. Then

$$\sqrt{X} = 16^{P} \cdot 4^{Q} \cdot \sqrt{M}$$

$$= 16^{P+Q} \cdot \sqrt{M*16^{-Q}}.$$

For a first approximation, we take

$$y_0 = A*(M+B) \cdot 16^{P} \cdot 4^{Q+1}$$

(Continued on next page)

DSQRT

Algorithm (Con't)

where the values of the constants are:

A = X'40385F07' = .22020005

B = X'40423A2A' = .25870006

This calculation is carried out with the characteristic of A increased by 8 and the others decreased by 8, in order to store the value of

$$(B \cdot 16^{P+Q}) \cdot 16^{-8}$$

for later use.

Then, two passes of the Newton-Raphson iteration are performed in single precision. The form of the iteration is:

$$y_{k+1} = \underbrace{\frac{1}{2}\left(\frac{x}{y_3} - y_3\right) + (B \cdot 16^{P+Q-8})}_{\text{single precision}} - (B \cdot 16^{P+Q-8}) + y_3$$

This is done to truncate excess digits of $\frac{x}{y_3} - y_3$, which is 0, and is less than 16^{P+Q-8} in absolute value.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DTAN Size of Code Area: 164 Hw

Stack Requirement: 30 Hw Data CSECT Size: 4 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DTAN

Function:
Computes tangent in double precision.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form: TAN(X), where X is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 302.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	radians

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
11	Argument outside range $\|X\| < \pi \cdot 2^{50}$	Return 1
12	Argument too near a singularity of the tangent function	Return maximum positive floating point number

Comments:
Error gets very large near a singularity, before error #12 is sent.
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Multiply X by $\frac{4}{\pi}$, and give the characteristic of this to X'0000000000000008' for use as a comparand to determine nearness to a singularity. The integer part of $|X \cdot \frac{4}{\pi}|$ is the octant. If the octant is even, let w = fraction part of $|X| \cdot \frac{4}{\pi}$.

(Continued on next page)

DTAN

Algorithm (Con't)

If the octant is odd, let w = -1 -fraction part of $|x| \cdot \frac{4}{\pi}$.

Next, compute two polynomials P(w) and Q(w). If $w \geq 2^{-46}$, then the forms of the polynomials are:

$$P(w) = w(a_0 + a_1 w^2 + a_2 w^4 + w^6)$$

$$Q(w) = b_0 + b_1 w^2 + b_2 w^4 + b_3 w^6$$

If $w < 2^{-46}$, then with u = w if $|x| \cdot \frac{4}{\pi} < 1$, and u = -w otherwise.

$$P(w) = w(a_0 + u)$$

$$Q(w) = b_0 + b_3 u$$

where the values of the constants are:

a_0 = X'C58AFDD0A41992D4' = -569309.04006345

a_1 = X'44AFFA6393159AA6' = 45050.3889630777

a_2 = X'C325FD4A87357CAF' = -607.8306953515

b_0 = X'C5B0F82C871A3B68' = -724866.7829840012

b_1 = X'4532644B1E45A133' = 206404.6948906228

b_2 = X'C41926DBBB1F469B' = -6438.8583240077

b_3 = X'422376F171F72282' = 35.4646216610

If w $\leq$ the comparand derived earlier and the octant = 1 or 2 (mod 4), then error 12 is sent. Otherwise, Q(w)/P(w) is returned with its sign adjusted. In octants = 0 or 3 (mod 4), P(w)/Q(w) is returned, with the sign adjusted according to tan(-x) = -tan(x).

The justification for this computation is that $\frac{P(w)}{Q(w)} = \tan(W \cdot \frac{\pi}{4})$ and $\frac{Q(w)}{P(w)} = \cot(w \cdot \frac{\pi}{4})$, and simple trigonometric identities give, for R = fraction part of X $\cdot \frac{4}{\pi}$:

(Continued on next page)

DTAN

Algorithm (Con't)

Octant (mod 4)	Formula for tan
0	$\tan(\|x\|) = \tan(R \cdot \frac{\pi}{4})$
1	$\tan(\|x\|) = \cot((1-R) \cdot \frac{\pi}{4})$
2	$\tan(\|x\|) = -\cot(R \cdot \frac{\pi}{4})$
3	$\tan(\|x\|) = -\tan((1-R) \cdot \frac{\pi}{4})$

which is the result of the computation as performed.

DTANH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DTANH Size of Code Area: 94 Hw

Stack Requirement: 22 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: DEXP

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DTANH

Function:
Computes hyperbolic tangent in double precision.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form: TAN(X), where X is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): (See below)

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: If $|X| > 20.101$, return +1 or -1, according to the sign of X.

If $0.54931 \leq |X| \leq 20.101$, then (using DEXP), $\tanh(|x|) = 1 - \frac{2}{1+e^{2|x|}}$

Restore sign with $\tanh(-x) = -\tanh(x)$. For $|X| \leq 16^{-7}$, $\tanh(x) = X$.

(Cont'd. on next page)

Execution Time:

$|X| \leq 3.72529E\text{-}9$: 47.8

$3.72529E\text{-}9 < |X| \leq 5.4931E\text{-}1$: 177.9

$5.4931E\text{-}1 < |X| < 2.0101E+1$: 420.6

$2.0101E + \leq |X|$: 54.6

DTANH

Algorithm (Con't)

For other values of X, $16^{-7} < |x| < 0.54931$, use a continued fraction approximation:

$$\tanh(x) = x + x^3 \left(\frac{C_0}{x^2 + C_1 + \frac{C_2}{x^2 + C_3 + \frac{C_4}{x^2 + C_5}}} \right)$$

where the values of the constants are:

C_0 = X'C0F6E12F40F5590A' = -.9643735440816707

C_1 = X'419DA5D6FD3DBC84' = 9.8529882328255392

C_2 = X'C31C504FEF537AF6' = -453.01951534852503

C_3 = X'424D2FA31CAD8D0C' = 77.186082641955181

C_4 = X'C3136E2A5891D8E9' = -310.8853383729134

C_5 = X'4219B3ACA4C6E790' = 25.701853083191565

EATAN2

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EATAN2 Size of Code Area: 132 Hw

Stack Requirement: 18 Hw Data CSECT Size: 10 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EATAN2

Function: Computes arctangent by fractional approximation in the range $(-\pi, \pi)$ in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ARCTAN2(X,Y), where X and Y are single precision scalars corresponding to sine and cosine respectively of the intended arctangent argument.

☐ Other Library Modules:

Execution Time (microseconds): 120.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar (sin)	SP	F0	-
Scalar (cos)	SP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
62	arg 1 = arg 2 = 0	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: The pointer to the data area that contains quadrant section constants is set and the sign of sin x is saved. The value $Z = |\sin x|/|\cos x|$ is checked for several special cases. (1) If cos x < 0 and $Z < (16)^{-6}$, then return $\pi \cdot$ SIGNUM(SINX). (2) If SINX = COSX = 0, then signal error and return 0. (3) If SINX ≠ 0, COSX = Ø, then return $\pm \pi/2 \cdot$ SIGNUM(Sin X). (4) If sin x ≠ 0, cos x ≠ 0, but $Z > (16)^{6}$, again return $\pm \pi/2 = (\pi/2 \cdot$ SIGNUM(sin x)). (5) If $Z < (16)^{-3}$, return Z.

Now, all of the special cases have been checked for. If the routine gets this far, it is time to reduce Z = tan x so that Z < tan π/12(tan 15°).

(Continued on next page)

EATAN2

Algorithm (Con't)

There are four cases to examine for Z in the 1st quadrant.

A) Z > 1. Use 1/Z.

1) tan 15° < 1/Z ≤ 1

2) 1/Z ≤ tan 15°

B) Z ≤ 1

3) tan 15° < Z ≤ 1

4) Z ≤ tan 15°

For Z or 1/Z > tan 15°, the reduction

$\text{Tan}^{-1}Z = \pi/6 + \tan(Y)$, where $Y = Z\sqrt{3} - 1/Z + \sqrt{3}$ is used.

To protect significant bits, Y is computed as

$Y = Z(\sqrt{3} - 1) - 1 + Z - 1/Z + \sqrt{3}$

Once Z or 1/Z ≤ tan 15°, the formula for arctan Z can be applied.

$\frac{\text{Tan}^{-1}(Z)}{Z} = D + CZ^2 + (B/(Z^2 + A))$, where the constants have the following values (hex values are used in the routine):

A = X'41168A5E' (1.4087812)

B = X'408F239C' (0.55913711)

C = X'BFD35F49' (-0.051604543)

D = X'409A6524' (0.60310579)

To adjust the angle to the proper section, the appropriate section constant is added to or subtracted from the intermediate result, as follows:

Z ≤ tan 15°	→	+ 0 (E'0')
tan 15° < Z ≤ 1	→	+ π/6 (X'40860A92')
1/Z ≤ tan 15°	→	- π/2 (X'C11921FB')
tan 15° < 1/Z ≤ 1	→	- π/3 (X'C110C152')

(Continued on next page)

EATAN2

Algorithm (Con't)

We now have the correct angle for the first quadrant. All that remains is to fix the quadrant. If the cos x < 0, then $Tan^{-1}(X) = \pi - Tan^{-1}(Z)$. That fixes the angle to agree with the sign of cos x. Now make the sign of the answer agree with the sign of sin x, i.e. $-Tan^{-1}(Z)$ for -sin x and $+ Tan^{-1}(Z)$ for + sin x.

The result, in radians, is in the correct quadrant in the range $(-\pi, +\pi)$.

Secondary Entry Name: ATAN

Function: Computes arctangent by fractional approximation in the range $(-\pi/2, +\pi/2)$ in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: ARCTAN(X), where X is a single precision scalar.

☐ Other library modules:

Execution Time (microseconds): 116.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Very similar to EATAN2, but the only special case that can be checked is Z = $|\tan X| < (16)^{-3}$. If Z is this small, then return Z to avoid an underflow exception later on. The algorithm for reduction and computation of Tan^{-1} Z is the same as EATAN2 again until quadrant fixing time. Since ARCTAN has only one arg, the result can only be adjusted in the range $(-\pi/2, \pi/2)$. The Tan^{-1} Z is computed for the first quadrant.

If the argument, tan x, is negative, the result is made negative.

EPWRE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EPWRE Size of Code Area: 32 Hw

Stack Requirement: 22 Hw Data CSECT Size: 4 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: EXP, LOG, SQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EPWRE

Function:
Exponentiation of a single precision scalar to a single precision scalar power.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
X**Y, where X and Y are single precision scalars.

☐ Other Library Modules:

Execution Time (microseconds): If Y = .5: 124.7
If Y ≠ .5: 337.1

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	SP	F0	-
Scalar (exponent)	SP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power ≤ 0	Return 0.0
24	Base < 0	use |base|

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: If exponent = 0.5 compute $X^{0.5}$ as $\sqrt{X}$.

Otherwise, $X^Y = e^{Y \log X}$, using the external EXP and LOG functions.

EPWRI

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EPWRI Size of Code Area: 38 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EPWRI

Function: Exponentiation of a single precision scalar to a double precision integer power.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
X**I, where X is a single precision scalar, and I is a double precision integer.

☐ Other Library Modules:

Execution Time (microseconds): (See next page)

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	SP	F0	-
Integer (exponent)	DP	R6	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power ≤ 0	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm: Let I = |exponent|, E = base. Write

$$I = \Sigma_{i=0}^{32} e_i 2^i, \quad \text{where } e_i = 0 \text{ or } 1 \text{ for all } i.$$

then

$$E^I = E^{\Sigma_i e_i 2^i} = \pi_i E^{e_i 2^i} = \pi_{e_i=1} E^{2^i} \quad \text{if some } e_i = 1, \text{ and } = 1 \text{ otherwise.}$$

(Continued on next page)

EPWRI

Execution Time:

38.2 + (n-1) 16.2 + 6.0m + 14.2 (if exponent negative),

where n = number of significant digits in binary representation of |exponent|.

m = number of significant 1's in binary representation of |exponent|.

Algorithm (Con't)

The product $\pi_{e_i=1} E^{2^i}$ is computed in a loop. Each time around the loop, E^{2^k} is multiplied by itself to give $E^{2^{k+1}}$. The k+1-st bit is shifted out of the exponent. If it is 1, $E^{2^{k+1}}$ is multiplied into the result. If not, the result is left alone. When the remaining exponent is zero, the loop is finished and the result is E^I. If the exponent was positive, return E^I. Otherwise, return the reciprocal of E^I.

Secondary Entry Name: EPWRH

Function:
Exponentiation of a single precision scalar to a single precision integer power.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
X**I, where X is a single precision scalar, and I is a single precision integer.

☐ Other library modules:

Execution Time (microseconds): Same as EPWRI, except constant is 38.8.

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	SP	F0	-
Integer (exponent)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power $\leq$ 0	Return 0.0

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm: Halfword exponent is shifted right to convert to a fullword. Then, EPWRI routine is used.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EXP Size of Code Area: 108 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EXP

Function:

Computes e^X in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

EXP(X), where X is a single precision scalar.

☒ Other Library Modules:

TANH, EPWRE, SINH

Execution Time (microseconds): 141.8

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
6	Argument outside range: $X \leq 174.673$	Return maximum positive floating point number.

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm: Let $X \log_2 e = 4R-S-T$, where R and S are integers, $0 \leq S \leq 3$, and $0 \leq T < 1$. Then

$$\exp(X) = 16^R \cdot 2^{-S} \cdot 2^{-T}$$

2^{-T} is computed by a fractional approximation of the form:

$$2^{-T} = 1 + \frac{2T}{CT^2 - T + D + \frac{B}{A + T^2}}$$

The computation is carried out in fixed-point, and the values and scaling of the constants are:

(Continued on next page)

EXP

Algorithm (Con't)

A = X'576AE119' = 87.417497 at bit 7

B = X'269F8E6B' = 617.97227 at bit 11

C = X'B9059003' = -0.03465736 at bit (-4)

D = X'B05CFCE3' = -9.95459578 at bit 4

The multiplication by 2^{-S} is carried out by shifting right S places, and the multiplication of 16^R is done by adding R to the floating exponent.

IPWRI

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IPWRI Size of Code Area: 46 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IPWRI

Function:

Computes double precision integer to positive double precision integer power.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

arg 1 ** arg 2, where arg 1 is a double precision integer, and arg 2 is a positive double precision literal.

☐ Other Library Modules:

Execution Time (microseconds): $k + 16.4(n-1) + 7.0m + 0.4(n-2)$ if $n \geq 2$, where $k = 44.6$, n = # of significant digits in binary representation of arg2, m = # of significant ones in binary representation of arg2.

Input Arguments:

Type	Precision	How Passed	Units
Integer(base)	DP	R5	-
Integer(exponent)	DP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
4	zero raised to power ≤ 0	Return 0

Comments:

Registers Unsafe Across Call: R5.

Algorithm: Shift all halfwords to convert to fullwords. Let B = base, I = exponent.

Write $I = \sum_{i=0}^{32} e_i 2^i$, where $e_i = 1$ for each i. Then:

$$B^I = \pi_i B^{e_i 2^i} = \pi_{e_i=1} B^{2^i} \text{ if } e_i = 1, \text{ and } = 1 \text{ otherwise.}$$

(Continued on next page)

IPWRI

Algorithm (Con't)

The product $\pi_{e_i=1} B^{2^i}$ is computed in a loop. Each time around the loop, B^{2^k} is multiplied by itself to give $B^{2^{k+1}}$. The k+1-st bit is shifted out of the exponent and tested. If it is 1, the partial result is multiplied by $B^{2^{k+1}}$. If not, the partial result is left as is. When the remaining exponent is 0, the result is B^I and the exit is taken from the loop. The answer is stored in ARG5 to be available after registers are restored.

Secondary Entry Name: IPWRH

Function:
Computes double precision integer to positive single precision integer power.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
arg 1 ** arg 2, where arg 1 is a double precision integer, and
arg 2 is a positive single precision integer literal.

☐ Other library modules:

Execution Time (microseconds): Same as for IPWRI, except k = 46.6.

Input Arguments:

Type	Precision	How Passed	Units
Integer(base)	DP	R5	-
Integer(exponent)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power <= 0	Return 0

Comments:
Registers Unsafe Across Call: R5.

Algorithm:
See IPWRI

Secondary Entry Name: HPWRH

Function:
Computes single precision integer to positive single precision integer power.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
arg 1 ** arg 2, where arg 1 is a single precision integer variable, and
arg 2 is a positive single precision positive integer literal.

☐ Other library modules:

Execution Time (microseconds): Same as for IPWRI, except k = 49.4.

Input Arguments:

Type	Precision	How Passed	Units
Integer(base)	SP	R5	-
Integer(exponent)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
4.	Zero raised to power <= 0.	Return 0.

Comments:
Registers Unsafe Across Call: R5.

Algorithm:

See IPWRI.

LOG

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: LOG Size of Code Area: 90 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: LOG

Function:
computes log(X) in single precision.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
LOG(X), where X is a single precision scalar.

☒ Other Library Modules: ASINH, ATANH, EPWRE, ACOSH

Execution Time (microseconds): 140.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
7	argument outside range X > 0	For X≤0, return LOG(|X|) for X = 0, return maximum negative floating point number.

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Write $X = 16^P \cdot 2^{-Q} \cdot M$, where P and Q are integers, $0 \le Q \le 3$, and $\frac{1}{2} \le M \le 1$. P, Q, and M are found by fixed-point calculations. Let A = 1, B = 0, if $M > \frac{\sqrt{2}}{2}$, and A = ½, B = 1 otherwise.

Let Z = (M-A)/(M+A). Then log(X) = (4P-Q-B)log 2 + log((1+Z)/(1-Z)).
Log((1+Z)/(1-Z)) is computed by an approximation of the form:

$$W + W\left(\frac{Rw^2}{S-W^2}\right)$$

where W = 2z, and the values of the constants are:

R = X'408D8BC7' = 0.55291413 S = X'416A298C' = 6.6351437

SIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: SIN Size of Code Area: 70 Hw

Stack Requirement: 0 Hw Data CSECT Size: 30 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: SIN

Function:
Computes sine(X) in single precision.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
SIN(X), where X is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): $-\pi \le X \le \pi$: 124.5
$X > \pi$ or $X < -\pi$: 123.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	radians

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
8	argument outside of range: $\|X\| < \pi \cdot 2^{18}$	return $\frac{\sqrt{2}}{2}$

Comments: Called as a library by compiler: uses only fixed-point registers R1 and R3, which are restored at exit from an intrinsic.
Registers Unsafe Across Call: R1,R3,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Let $|X| \cdot \frac{4}{\pi} = Q+R$, Q an integer, $0 \le R < 1$. Add 4 to Q if the sine is desired and X < 0, and add 2 to Q if the cosine is desired. Since sin(-x) = sin(x+π), and $\cos(x) = \sin(\frac{\pi}{2} + x)$. This reduces the problem of computing sin(x) for $X \ge 0$.

(Continued on next page)

SIN

Algorithm (Con't)

Since Q has been adjusted, it is only necessary to compute $\sin(\frac{\pi}{4}(Q+R))$.

If $Q_o = Q \bmod 8$, then this is equal to $\sin(\frac{\pi}{4}(Q_o+R))$. The eight cases of this yield, through simple trigonometric identities:

$Q_o = 0$: $\sin(R \cdot \frac{\pi}{4})$

1: $\cos((1-R) \cdot (\frac{\pi}{4})$

2: $\cos(R \cdot \frac{\pi}{4})$

3: $\sin((1-R) \cdot \frac{\pi}{4})$

4: $-\sin(R \cdot \frac{\pi}{4})$

5: $-\cos((1-R) \cdot \frac{\pi}{4})$

6: $-\cos(R \cdot \frac{\pi}{4})$

7: $-\sin((1-R) \cdot \frac{\pi}{4})$

Let $R_o = R$ in octants 0, 2, 4, 6 and $R_o = 1-R$ in octants 1, 3, 5, 7.

We compute $\sin(R_o \cdot \frac{\pi}{4})$ in octants 0, 3, 4, 7 and $\cos(R_o \cdot \frac{\pi}{4})$ in octants 1, 2, 5, 6, and negate the result in octants 4, 5, 6, 7.

$\sin(R_o \cdot \frac{\pi}{4})$ and $\cos(R_o \cdot \frac{\pi}{4})$ are computed by polynomial approximations.

SIN

Algorithm (Con't)

The form of the approximation for sine is:

$$\sin(R_0 \cdot \frac{\pi}{4}) = R_0(a_0 + a_1R_0^2 + a_2R_0^4 + a_3R_0^6)$$

where the values of the constants are:

a_0 = X'40C90FDB' = ,78539819

a_1 = X'C014ABBC' = -.080745459

a_2 = X'3EA32F62' = .0024900069

a_3 = X'BD25B368' = -.000035943

The form of the approximatin for cosine is;

$$\cos(R_0 \cdot \frac{\pi}{4}) = 1 + a_1R_0^2 + a_2R_0^4 + a_3R_0^6$$

where the values of the constants are:

a_1 = X'C04EF4EE' = -,30842483

a_2 = X'3F40ED0F' = .0158510767

a_3 = X'BE14F17D' = -.000319570

Secondary Entry Name: COS

Function:
Computes cosine(X) in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
COS(X), where X is a single precision scalar,

☐ Other library modules:

Execution Time (microseconds): $-\pi \leq X \leq \pi$: 122.1
$X > \pi$ or $X < -\pi$: 123.1

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	radians

Output Results:

Type	Precision	How Passed	Units
scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
8	argument outside range $\|X\| < \pi.2^{18}$	Return $\frac{\sqrt{2}}{2}$

Comments:

See SIN Comments.
Registers Unsafe Across Call: R1,R3,R4,F0,F1,F2,F3,F4,F5.

Algorithm:

See SIN Algorithm.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: SINH Size of Code Area: 80 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: EXP

ENTRY POINT DESCRIPTIONS

Primary Entry Name: SINH

Function:
Computes hyperbolic sine in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
SINH(X), where X is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): $1 \leq |X| \leq 1.75366E+2$: 235.6
$2.0394E-4 \leq |X| < 1$: 80.7
$|X| < 2.0394E-4$: 40.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
9	Argument outside range $\|X\| \leq 175.366$	Return maximum positive floating point number.

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: If $X < 2.04E-4$, then $\sinh(x) = x$. If $2.04E-4 \leq |X| < 1$, then $\sinh(x)$ is computed via a polynomial approximation.

The form of the polynomial is:

$$\sinh(x) = X + C_1X^3 + C_3X^5 + C_3X^7$$

where the values of the constants are:

(Continued on next page)

SINH

Algorithm (Con't)

C_1 = X'402AAAB8' = 0.16666734

C_2 = X'3F221E8C' = 0.008329912

C_3 = X'3DD5D8B3' = .2039399E-3

Otherwise, sinh(|X|) or cosh(|X|) is calculated using EXP. The number V, equal to 0.4995050, is introduced to control rounding errors and the formula is as follows:

$$\sinh(x) = \frac{1}{2V}\left(e^{(x+\log v)} - \frac{v^2}{e^{(x+\log v)}}\right)$$

$$\cosh(x) = \frac{1}{2V}\left(e^{(x+\log v)} + \frac{v^2}{e^{(x+\log v)}}\right)$$

the identities sinh(-x) = -sinh(x) and cosh(-x) = cosh(x) are used to recover sinh(x) and cosh(x) from sinh(|x|) and cosh(|x|).

Secondary Entry Name: COSH

Function:

Computes hyperbolic cosine in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: SINH(X), where X is a single precision scalar.

☐ Other library modules:

Execution Time (microseconds): 228.9

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
9	Argument outside range $\|x\| \leq 175.366$	Return maximum positive floating point number.

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

See SINH algorithm.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: SQRT Size of Code Area: 48 Hw

Stack Requirement: 0 Hw Data CSECT Size: 14 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: SQRT

Function:

Computes square root in single precision.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:
SQRT(X), where X is a single precision scalar.

[x] Other Library Modules: ACOS, ASINH, EPWRE, VV10S3 , ACOSH

Execution Time (microseconds): 88.3

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
5	Argument outside range $X \geq 0$	Return SQRT ($\|X\|$)

Comments:

Registers Unsafe Across Call: R1, R4, R5, R6, R7, F0, F1, F2, F3.

Algorithm: Write $X = 16^{2P-Q} \cdot M$, where $\frac{1}{16} \leq M < 1$. Then, $\sqrt{X} = 16^{P} \cdot 4^{-Q} \cdot \sqrt{M}$.

This fact is used to obtain a good first approximation to $\sqrt{X}$ by approximating $\sqrt{M}$ by a hyperbolic approximation.

(Continued on next page)

SQRT

Algorithm (Con't)

The form of the approximation is, for Q=0

$$\sqrt{M} = a - \frac{b}{(\frac{C}{2}) + (\frac{M}{2})} \qquad [\frac{C}{2} + \frac{M}{2} \text{ is to avoid fixed-point overflow for large M}]$$

where the calculations are done in fixed-point.

The values of the constants are:

a = X'01AE7D00' = 1.6815948 at bit 7

b = X'FF5B02F1' = -1.2889728 at bit 7

$\frac{C}{2}$ = X'35CFC610' = 0.42040325 at bit 0

For Q = 1, $\frac{a}{4}$ and $\frac{b}{4}$ are used instead of a and b.

$\frac{a}{4}$ = X'006B9F40' = 0.4203987 at bit 7

$\frac{b}{4}$ = X'FFD6C0BD' = -0.3222432 at bit 7

The first approximation is improved with two passes of the Newton-Raphson iteration. The form of the first is:

$$y_1 = \frac{1}{2}(\frac{x}{y_0} + y_0)$$

The form of the second, to minimize truncation errors, is:

$$y_2 = \frac{1}{2}(\frac{x}{y_0} - y_0) + y_0$$

y_2 is returned as the answer.

TAN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: TAN Size of Code Area: 112 Hw

Stack Requirement: 20 Hw Data CSECT Size: 4 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: TAN

Function:
Computes tangent in single precision.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
TAN(X), where X is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 164.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	radians

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
11	Argument outside range $\|x\| < \pi \cdot 2^{18}$	Return 1
12	Argument too close to singularity of tangent function	Return maximum positive floating point number

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Let $|x| \cdot \frac{4}{\pi} = Q+R$. Q an integer, $0 \leq R < 1$. Give the characteristic of $|x| \cdot \frac{4}{\pi}$ to X'00000008' for later use as a comparand, to determine nearness of X to a singularity.

(Continued on next page)

TAN

Algorithm (Con't)

We have the following table:

Q mod 4	tan(x)
0	$\tan(\|x\|) = \tan(R \cdot \frac{\pi}{4})$
1	$\tan(\|x\|) = \cot((1-R) \cdot \frac{\pi}{4})$
2	$\tan(\|x\|) = -\cot(R \cdot \frac{\pi}{4})$
3	$\tan(\|x\|) = -\tan((1-R) \cdot \frac{\pi}{4})$

For Q mod 4 even, let w = R, and for Q mod 4 odd, let w = 1-R. Compute two polynomials in w, as polynomials in

$u = \frac{w^2}{2}$:

$$P(w) = w(a_0 + u)$$

$$Q(w) = b_o + b_1 u + b_2 u^2$$

then $\tan(w) = \frac{P(w)}{Q(w)}$, $\cot(w) = \frac{Q(w)}{P(w)}$, and the above table describes how tan(x) is computed. Finally, tan(x) is computed using the identity tan(-x) = -tan(x).

The values of the constants are:

a_0 = X'C1875FDC' = -8.4609032

b_0 = X'C1AC5D33' = -10.7727537

b_1 = X'415B40FD' = 5.7033663

b_2 = X'C028C93F' = -.15932077

TANH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: TANH Size of Code Area: 56 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: EXP

ENTRY POINT DESCRIPTIONS

Primary Entry Name: TANH

Function:
Computes hyperbolic tangent in single precision.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
TANH(X), where X is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): $|X| \leq 2.4414E\text{-}4$: 38.2
$2.4414E\text{-}4 < |X| \leq 7.0E\text{-}1$: 78.7, $7.0E\text{-}1 < |X| < 9.011$: 224.4
$9.011 \leq |X|$: 42.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: If $|X| > 9.011$, return +1 or -1, according to the sign of X. If $0.7 \leq |X| \leq 9.011$, then (using EXP), $\tanh(|x|) = 1-2/(1+e^{2|x|})$. Restore sign with $\tanh(-x) = -\tanh(x)$. For $|X| \leq 16^{-3}$, $\tanh(x) = X$.

For other values of X, $16^{-3} < |X| < 0.7$, use a rational approximation.

where the values of the constants are:

a = X'BEF7EA70' = -.003782895

b = X'C0D08756' = -.81456511

c = X'41278C49' = 2.4717498

5.3.3 Vector/Matrix Routine Descriptions

This subsection presents a class of routines which deal with HAL/S vector/matrix operations. Some of the routines implement HAL/S language built-in functions while others implement HAL/S operators.

MM0DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM0DNP **Size of Code Area:** 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: none.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM0DNP

Function:
Moves a double precision scalar to all positions in an n x m partition of a double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$M^{*}_{A \text{ TO } B,C \text{ TO } D} = 0$; where M is a double precision matrix.

☐ Other Library Modules:

Execution Time (microseconds): 6.8 + n(4.0 + 8.0m)

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix (n,m)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1, R3, R4, R5, R6, R7, F0, F1.

Algorithm: Uses two nested loops:
Outer loop selects row;
Inner loop selects column and moves scalar to current row/column position.
Upon exiting inner loop, 'outdel' is added to pointer to output matrix location.

MM0SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM0SNP **Size of Code Area:** 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM0SNP

Function:
Fill an n x m partition of a single precision matrix with a single precision scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$M^{*}_{A \text{ TO } B,C \text{ TO } D} = 0$; where M is a single precision matrix.

☐ Other Library Modules:

Execution Time (microseconds): 6.4 + n(4.4 + 6.4m)

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Integer(n)	SP	R5	-
Intger(m)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix (n,m)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1, R3, R4, R5, R6, R7, F0, F1.

Algorithm: Uses two nested loops, one on n; one on m;
Inner loop selects row and column of result matrix and moves input scalar into location.
At exit of inner loop, pointer to matrix element is incremented by outdel, new row is selected, and inner loop is executed again.

MM1DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM1DNP Size of Code Area: 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM1DNP

Function: Moves a partition of a matrix in double precision.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
$M2=M1_{A\ TO\ B,C\ TO\ D}$; where M1 and M2 are double precision matrices.
$M2_{A\ TO\ B,C\ TO\ D}=M1$;

[ ] Other Library Modules:

Execution Time (microseconds): 10.8 + n(5.4 + 12.2m) for n x m matrix moved.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R2 → 0th element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel,outdel)	DP	R7(indel in highest HW, outdel in Low HW)	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3.

Algorithm:
```
Loops on # rows;
   Loops on # columns;
      Load and Store current element pointed to by input/output pointers;
      Increment pointers to next row element;
   End column loop;
   Increment input pointer by indel;
   Increment output pointer by outdel;
End row loop;
```


MM1SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM1SNP Size of Code Area: 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM1SNP

Function: Moves a partition of a single precision matrix.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
$M2 = M1_{A\ TO\ B,\ G\ TO\ D}$; where M1 and M2 are single precision
$M2_{A\ TO\ B,\ C\ TO\ D}=M1$; matrices.

[ ] Other Library Modules:

Execution Time (microseconds): 10.8 + n(5.4 + 9.4m) for n x m matrix.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R2 → 0th element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel, outdel)	DP	R7(high HW=indel,Low HW=outdel)	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm:
```
Loop on # rows;
   Loop on # columns;
      Load and store current element point to by input/output pointer;
      increment pointers to next row;
   End column loop;
   Increment input pointer by indel;
   Increment output pointer by outdel;
End row loop;
```

MM1TNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM1TNP **Size of Code Area:** 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM1TNP

Function:
Moves a partition of a double precision matrix and stores it as single precision.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**

$M2=M1_{A\ TO\ B,\ C\ TO\ D}$; where M2 is a single precision matrix.

$M2_{A\ TO\ B,\ C\ TO\ D}$; where M1 is a double precision matrix.

☐ **Other Library Modules:**

Execution Time (microseconds): 10.4 + n(5.8 + 10.6m) for n x m move.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R2 → 0th element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel,outdel)	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3.

Algorithm:
```
Loops on # rows;
  Loops on # columns;
    Load long input element pointed to by input pointer;
    Store short into output element pointed to by output pointer;
    Increment pointer to next element;
  End column loops;
  Increment input pointer by indel;
  Increment output pointer by outdel;
End row loop;
```


MM1WNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM1WNP **Size of Code Area:** 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM1WNP

Function: Moves a partition of a single precision matrix and stores it as a double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$M2=M1_{A\ TO\ B,\ C\ TO\ D}$; where M1 is a single precision matrix;

$M2_{A\ TO\ B,C\ TO\ D}=M1$; M2 is a double precision matrix.

☐ Other Library Modules:

Execution Time (microseconds): 13.6 + n(5.0 + 11.0m) for n x m move.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R2 → 0th element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel,outdel)	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3.

Algorithm:
```
Clears lower half of floating point register pair.
Loops on # rows;
  Loops on # columns;
    Load short input element pointed to by input pointer;
    Store long (with zeroed second word) into output element
          pointer;
    Increment pointers to next row element;
  End column loop;
  Increment input pointer by indel;
  Increment output pointer by outdel;
End row loop;
```

MM6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM6DN **Size of Code Area:** 42 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM6DN

Function: Multiplies two double precision matrices,

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:

M1 M2, where M1 is a m x n double precision matrix
M2 is a n x ℓ double precision matrix } m,n,$\ell \neq 3$.

[ ] Other Library Modules:

Execution Time (microseconds): 22.2 + m(10.8 + ℓ(21.2 + 27.2n))

Input Arguments:

Type	Precision	How Passed	Units
matrix (m,n)	DP	R2 → 0th element	-
matrix (n,ℓ)	DP	R3 → 0th element	-

(Continued on next page)

Output Results:

Type	Precision	How Passed	Units
Matrix (m,ℓ)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm: Uses 3 loops:
Innermost (on n) multiplies a row of M1 by a column of M2;
The second loop (on ℓ) resets the column pointer;
The outer loop (on m) resets the row pointer.

MM6DN

Input Arguments (Con't):

Type	Precision	How Passed	Units
Integer (m)	SP	R5	-
Integer (n)	SP	R6	-
Integer (ℓ)	SP	R7	-

MM6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM6D3 Size of Code Area: 32 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM6D3

Function: Multiplies two 3 x 3 double precision matrices.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
M1 M2, where M1 and M2 are double precision 3 x 3 matrices.

☐ Other Library Modules:

Execution Time (microseconds): 671.6

Input Arguments:

Type	Precision	How Passed	Units
Matrix (M1)	DP	R2 → 0th element	-
Matrix (M2)	DP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix (3,3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm: Explicitly multiplies row by column, element by element. Uses BCTB to advance to each new col., and BCTB to advance to each new row.

MM6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM6SN Size of Code Area: 40 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM6SN

Function: Multiplies two single precision matrices.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
M1 M2, where M1 is a m x n single precision matrix, M2 is a n x ℓ single precision matrix } $m,n,\ell \neq 3$.

☐ Other Library Modules:

Execution Time (microseconds): $22.2 + m(10.8 + \ell(20.2 + 18.0n))$.

Input Arguments:

Type	Precision	How Passed	Units
Matrix (m,n)	SP	R2 → 0th element	-
Matrix (n,ℓ)	SP	R3 → 0th element	-

(Continued on next page)

Output Results:

Type	Precision	How Passed	Units
Matrix (m,ℓ)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm: Same as MM6DN.

MM6SN

Input Arguments (Con't):

Type	Precision	How Passed	Units
Integer (m)	SP	R5	-
Integer (n)	SP	R6	-
Integer (ℓ)	SP	R7	-

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM6S3 Size of Code Area: 24 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM6S3

Function: Multiplies two 3 x 3 single precision matrices.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:
M1 M2, where M1 and M2 are 3 x 3 single precision matrices.

[ ] Other Library Modules:

Execution Time (microseconds): 409.6

Input Arguments:

Type	Precision	How Passed	Units
Matrix (M1)	SP	R2 → 0th element	-
Matrix (M2)	SP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix (3,3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm: Same as MM6D3

MM11DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM11DN Size of Code Area: 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: _____

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM11DN

Function:

Transposes an n x m double precision matrix.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:

$TRANSPOSE(M^{*})$ or M^{*T}, where M is an n x m double precision matrix, and m and/or n≠3.

[ ] Other Library Modules:

Execution Time (microseconds): 8.0 + m(5.8 + 12.2n)

Input Arguments:

Type	Precision	How Passed	Units
Matrix (n,m)	DP	R2 → 0th element	-
Integer	SP	R5	-
Integer	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Matrix (m,n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3.

Algorithm: Uses two nested loops:
Outer loop selected column of input matrix;
Inner loop moves elements of selected column to corresponding row of result matrix.

MM11D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM11D3 Size of Code Area: 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM11D3

Function: Performs transpose of 3x3 double precision matrix.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:

$TRANSPOSE(M)$ or M^{*T}, where M is a 3x3 double precision matrix.

[ ] Other Library Modules:

Execution Time (microseconds): 93.6

Input Arguments:

Type	Precision	How Passed	Units
matrix (3,3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
matrix (3,3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3,F4,F5.

Algorithm: Uses loop to load elements of one column into registers, then store into row elements of resultant for each pass through the loop.

MM11SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM11SN Size of Code Area: 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM11SN

Function:
Transposes an n x m single precision matrix.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
$TRANSPOSE(M^{*})$
or M^{*T} , where M is an n x m single precision matrix and m and/or n ≠ 3.
☐ Other Library Modules:

Execution Time (microseconds): 8.4 + m(5.8 + 9.4n)

Input Arguments:

Type	Precision	How Passed	Units
Matrix (n,m)	SP	R2 → 0th element	-
Integer (# rows=n)	SP	R5	-
Integer (# columns=m)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Matrix(m,n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm: Uses two nested loops:
Outer loop selects which column of input matrix to use;
Inner loop loads and stores column elements as row elements of result.

MM11S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM11S3 Size of Code Area: 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM11S3

Function: Performs transpose of 3x3 single precision matrix.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
$TRANSPOSE(M^{*})$, where M is a 3x3 single precision matrix.
or M^{*T}
☐ Other Library Modules:

Execution Time (microseconds): 71.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm: Uses loop to load F0, F2, F4 with columns of input matrix and store them as rows of output matrix for columns 1 → 3, rows 1 → 3.

MM12DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM12DN Size of Code Area: 150 Hw

Stack Requirement: 22 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM12DN

Function: Find the determinant of an n x n double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

D = DET(M), where D is a declared double precision scalar, and M an n x n, double precision matrix.

☐ Other Library Modules:

Execution Time (microseconds): for n = 2: 63.2
for n ≥ 4: (Continued on bottom of this page)

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R2 → 0th element	-
Matrix(n,n) workarea	DP	R4	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Same as MM12SN

Execution Time (Continued):

$$\text{minimum time} = 59.4 + 10.2n^2 + \sum_{k=1}^{n-1} (54.8k^2 + 81.2k + 115.6)$$

$$\text{Maximum time} = 59.4 + 10.2n^2 + \sum_{k=1}^{n-1} (60.2k^2 + 134.8k + 169.0 + 3.6n)$$

See MM12SN LRD for a description of maximum time vs. minimum time.

MM12D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM12D3 Size of Code Area: 44 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM12D3

Function:

Finds determinant of 3x3 double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overset{*}{DET}(M)$, where M is a double precision 3x3 matrix.

☒ Other Library Modules: MM14D3

Execution Time (microseconds): 229.6

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Uses direct code, no loops to calculate determinant. See algorithm for MM12S3.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM12SN Size of Code Area: 1308 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM12SN

Function: Finds the determinant of an n x n single precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

S = DET(M); where S is declared a single precision scalar, and M is an n x n single precision matrix.

☐ Other Library Modules:

Execution Time (microseconds): for n = 2: 44.4
for n ≥ 4: (Continued on next page)

Input Arguments:

Type	Precision	How Passed	Units
Matrix	SP	R2 → 0th element	-
Matrix(n,n) workarea	SP	R4	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

```
DET = 1.0
FOR K = 1 TO N1 DO
   BIG = 0
   I1 = J1 = K
   FOR I = K TO N DO
   FOR J = K TO N D)
     IF ABS(A(I,J)) > BIG THEN
        BIG = ABS(A(I,J));
        I1 = I;
        J1 = J;
   IF I1 ≠ K THEN
```

find maximal element

(Continued on next page)

MM12SN

Execution Time (microseconds) (Continued):

$$\text{minimum time} = 47.8 + 7.8n^2 + \sum_{k=1}^{n-1} (37.6k^2 + 64.6k + 85.8)$$

$$\text{maximum time} = 47.8 + 7.8n^2 + \sum_{k=1}^{n-1} (41.6k^2 + 105.8k + 127.2 + 3.6n)$$

The minimum time occurs in the event that all matrix elements are positive and where no row or column switching is required at any point of the computation.

The maximum time occurs in the event that all matrix elements require complementing to obtain their absolute value, BIG changes on every comparison, and row and column switching are required at every point in the computation.

Algorithm (Con't)

```
      DET = -DET
      FOR J = K TO N SWITCH(A(I1,J),A(K,J)      switch rows
   IF J1 ≠ K THEN
      DET = -DET
      FOR I = K TO N SWITCH(A(I,J1),A(JK);      switch columns.
      DET = DET*A(K,K)                  product of diagonal element
      FOR I = K + 1 TO N DO
        TEMP1 = -A(1,K)/A(K,K)                  reduce
        FOR J = K + 1 TO N DO
            A(I,J) = A(I,J) + A(K,J) * TEMP1
   DET = DET*A(N,N)              last diagonal element
```

If dim = 2, then special case:

```
DET = A(1,1)*A(2,2)-A(1,2)*A(2,1)
```

MM12S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM12S3 Size of Code Area: 26 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM12S3

Function:

Finds determinant of a single precision 3x3 matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\det^{*}(M)$, where M is a 3x3 single precision matrix.

☒ Other Library Modules: MM14S3

Execution Time (microseconds): 116.0

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Uses direct inline code to calculate

$$\det = M_{11} M_{22} M_{33} + M_{12} M_{23} M_{31} + M_{13} M_{21} M_{32}$$

$$-M_{31} M_{22} M_{13} - M_{32} M_{23} M_{11} - M_{33} M_{21} M_{12}.$$

MM13DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM13DN Size of Code Area: 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM13DN

Function:

Calculates TRACE of an n x n double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\mathrm{TRACE}^{*}(M)$, where M is an n x n double precision matrix, n≠3.

☐ Other Library Modules:

Execution Time (microseconds): 12.0 + 10.2n

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R2 → 0th element	-
Integer(n)	SP	R5	

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6,F0,F1.

Algorithm: Creates a skip value of n+1; Uses loop counting down n-1 to zero, each pass summing a diagonal element of the matrix by using the skip value to increment from the previous diagonal element.

MM13D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM13D3 Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM13D3

Function:

Calculates TRACE of a 3x3 double precision matrix.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:

$\text{TRACE}(\overset{*}{M})$, where M is a 3x3 double precision matrix.

[ ] Other Library Modules:

Execution Time (microseconds): 19.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R2,R4,F0,F1.

Algorithm: Direct code, no loops to calculate

$$M_{11} + M_{22} + M_{33}.$$

MM13SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM13SN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM13SN

Function:

Calculates TRACE of an n x n single precision matrix.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:

$\text{TRACE}(\overset{*}{M})$, where M is a single precision n x n matrix, $n \neq 3$.

[ ] Other Library Modules:

Execution Time (microseconds): 8.8 + 6.2n

Input Arguments:

Type	Precision	How Passed	Units
Matrix (n,n)	SP	R2 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6,F0,F1.

Algorithm: Creates a skip value of n+1; uses loop counting down n-1 to zero, each pass summing a diagonal element of the matrix by using the skip value to increment from the previous diagonal element.

MM13S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM13S3 Size of Code Area: 4 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM13S3

Function:

Calculates TRACE of a 3x3 single precision matrix.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:
TRACE(M), where M is a 3x3 single precision matrix.

[ ] Other Library Modules:

Execution Time (microseconds): 9.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix (3,3)	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R2,R4,F0,F1.

Algorithm:

Straight code to calculate $M_{11} + M_{22} + M_{33}$.

MM14DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM14DN Size of Code Area: 258 Hw

Stack Requirement: 20 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: MM15DN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM14DN

Function: Inverts an n x n double precision matrix.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:
M^{-1}, where M is an n x n double precision matrix, $n \neq 3$.
or INVERSE(M)

[ ] Other Library Modules:

Execution Time (microseconds): for n = 2: 173.8,
for $n \geq 4$: $63.0 + 129.5n + 43.0n^2 + 65.4n^3$

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R4 → 0th element	-
Integer(n)	SP	R5	-
Matrix(n,n) workarea	DP	R7	

Output Results:

Type	Precision	How Passed	Units
Matrix	DP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
27	Singular matrix	Return identity matrix

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Same as MM14SN, except that pivot element divide operation is done by multiplying by reciprocal to save time over use of long divide instruction.

MM14D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM14D3 Size of Code Area: 128 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: MM12D3 MM15DN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM14D3

Function:

Inverts a 3x3 double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

M^{*-1} or $INVERSE(M)^{*}$, where M is a 3x3 double precision matrix.

☐ Other Library Modules:

Execution Time (microseconds): 795.4

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R4 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
27	Attempted inverse of singular matrix.	Return identity matrix.

Comments: Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Explicit code, no loops; algorithm same as MM14S3 except that external routines used are MM12D3 and MM15DN.

MM14SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM14SN Size of Code Area: 242 Hw

Stack Requirement: 20 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: MM15SN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM14SN

Function: Inverts a single precision n x n matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

M^{-1} or $INVERSE(M)$, where M is a single precision n x n matrix, $n \neq 3$.

☐ Other Library Modules:

Execution Time (microseconds): for n = 2: 107.6,

for $n \geq 4$: $52.0 + 39.2n + 10.5n^2 + 54.6n^3$

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	SP	R4 → 0th element	-
Integer(n)	SP	R5	-
Matrix(n,n) workarea	SP	R7	

Output Results:

Type	Precision	How Passed	Units
Matrix	SP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
27	Matrix is singular.	Return identity matrix.

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:
```
For K = 1, N
    FIND MAXIMAL ELEment in row K to n, cols, K to n
    save it as 'big' pivot element
    save its row # as ISW(K)
    save its col # as JSW(K)
    switch Kth and ISW(K)th row
    switch Kth and JSW(K)th col
    divide Kth col except for Kth element by - BIG
    reduce matrix
    divide Kth row except for Kth element by big
    replace pivot by reciprocal
DO K = N-1, 1
    interchange JSW(K)th and Kth rows
    interchange ISW(K)th and Kth cols.
```

MM14S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM14S3 Size of Code Area: 80 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: MM12S3, MM15SN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM14S3

Function: Inverts a 3x3 single precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: M^{*-1} or INVERSE(M^{*}), where M is a 3x3 single precision matrix.

☐ Other Library Modules:

Execution Time (microseconds): 458.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix	SP	R4 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix	SP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
27	Attempted inverse of singular matrix.	Return an identity matrix.

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Explicit code, no loops to calculate:

$$\text{inverse } M = \frac{adj(M)}{|M|}, \text{ where } adjM_{i,j} = detM_{i\neq3,j\neq3} \text{ and } |M| = detM$$

uses external determinant routine (MM12S3) and in event of determinant of zero, calls identity matrix routine (MM15SN).

MM15DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM15DN Size of Code Area: 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM15DN

Function:

Creates an n x n double precision identity matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: M^{*0}, where M is an n x n double precision matrix.

☒ Other Library Modules: MM14DN, MM14D3

Execution Time (microseconds): $15.6 + 5.0n + 11.2n^2$

Input Arguments:

Type	Precision	How Passed	Units
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R4,R5,R6,R7,F0,F1,F2,F3.

Algorithm: Uses two nested loops, each counting 1 to n. Inner loop compares both loop indices; if equal, stores 1.0 at current row/column position; otherwise stores 0.0.

MM15SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM15SN Size of Code Area: 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: _____

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM15SN

Function:

Creates an n x m identity matrix.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
M^{*0}, where M is a single precision n x n matrix.

[X] Other Library Modules: MM14SN, MM14S3

Execution Time (microseconds): $10.0 + 5.2n + 9.6n^2$

Input Arguments:

Type	Precision	How Passed	Units
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Matrix (n,n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R4,R5,R6,R7,F0,F1,F2,F3.

Algorithm: Uses two nested loops, each counting 1 to n.
Inner loop checks both loop indices; if equal, stores
a 1.0 at current row/column position, otherwise stores 0.0.

MM17D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM17D3 Size of Code Area: 86 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM17D3

Function: Raises a 3 x 3 double precision matrix to a power.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
M^{*I}, where M is a 3x3 double precision matrix, and
I is an integer literal > 1.

[ ] Other Library Modules:

Execution Time (microseconds): (On next page).

Input Arguments:

Type	Precision	How Passed	Units
matrix(3,3)	DP	R4 → 0th element	-
integer(power)	SP	R6	-
Matrix(3,3) workarea	DP	R7	

Output Results:

Type	Precision	How Passed	Units
Matrix (3,3)	DP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Loads R5 with literal 3 and drops in MM17DN code.

MM17D3

Execution Time (microseconds):

Exponent = 2: 991.6

Exponent > 2: 1071.2 · (# of significant zeros in exponent)
+ 2137.2 · (# of ones in exponent)
- 2105.8

MM17D3

Secondary Entry Name: MM17DN

Function: Raises an n x n double precision matrix to a power.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
M^{**I}, where M is an n x n double precision matrix, and
I is an integer literal > 1.

[ ] Other library modules:

Execution Time (microseconds): (See below)

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R4 → 0th element	-
integer(n)	SP	R5	-
integer(power)	SP	R6	-
matrix(n,n) workarea	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix (n,n)	DP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Same as MM17SN.

Execution Time (microseconds):

$27.8n^3 + 19.4n^2 + 6.2n + 43.4$ if power = 2.

$124.2 + TMULT(KA) + TMOVE(KA-1) + 8.6\ KB + 3.4\ KC$ if power > 2.

where:

TMULT = $9.6 + 6.2n + 19.4n^2 + 27.8n^3$

KA = (((# significant 1's in exponent)-1)·2) + (# of significant 0's in exponent)

TMOVE = $10.2 + 11.0n^2$

KB = total number of significant 1's and 0's in exponent

KC = # of significant 1's in exponent.

MM17S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM17S3 Size of Code Area: 78 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM17S3

Function: Raises a 3x3 single precision matrix to a power.

Invoked by:

☑ Compiler emitted code for HAL/S construct of the form:
M^{*I}, where M is a 3x3 single precision matrix, and
I is an integer literal > 1.

☐ Other Library Modules:

Execution Time (microseconds): (On next page)

Input Arguments:

Type	Precision	How Passed	Units
matrix(3,3)	SP	R4 → 0th element	-
integer(power)	SP	R6	-
matrix(3,3) workarea	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix	SP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Loads R5 with literal 3 and drops into MM17SN code.

MM17S3

Execution Time (microseconds):

exponent = 2: 623.6

exponent > 2: 681.6 · (# significant zeros in exponents)
+ 1358.0 · (# ones in exponent)
- 1305.0.

MM17S3

Secondary Entry Name: MM17SN

Function: Raises an n x n single precision matrix to a power.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
M^{*I}, where M is an n x n single precision matrix, and
I is an integer literal > 1.
☐ Other library modules:

Execution Time (microseconds): (See below).

Input Arguments:

Type	Precision	How Passed	Units
matrix(n,n)	SP	R4 → 0th element	-
integer(n)	SP	R5	-
integer(power)	SP	R6	-
matrix(n,n) workareas	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix (n,n)	SP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Let A = original matrix, R = result matrix, T = temporary matrix.
1) R = A A
2) locate first one bit in exponent, remove it, remember bit position
3) go to step 6
4) T = R
5) R = T T
6) Remove exponent bit at current position, increment position. If bit was 0 go to step 9.
7) T = R
8) R = T A
9) If any bits left in exponent, go to step 4, otherwise R is complete.

Execution Time (microseconds):
if power = 2: then $15.6n^3 + 15.2n^2 + 5.8n + 43.8$
if power > 2: same as in MM17DN except
$TMULT = 10.0 + 5.8n + 15.2n^2 + 15.6n^3$
$TMOVE = 10.2 + 8.6n^2$

MV6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MV6DN Size of Code Area: 24 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MV6DN

Function: Multiplies a double precision mxn matrix by a length n double precision vector.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
$M^{*} \overline{V}$, where M is an mxn matrix by a length n,
V is a length n double precision vector.
☐ Other Library Modules:

Execution Time (microseconds): 12.0 + m(19.3 + 26.0n)

Input Arguments:

Type	Precision	How Passed	Units
Matrix(m,n)	DP	R2 →0th element	-
Vector(n)	DP	R3 →0th element	-
Integer(m)	SP	R5	-
Integer(n)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Vector(m)	DP	R1 →0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm: Uses 2 nested loop, outer loop selecting rows of matrix, inner loop summing products of vector elements with current row elements.

MV6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MV6D3 Size of Code Area: 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MV6D3

Function: Multiplies a 3x3 double precision matrix by a length 3 double precision vector.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overset{*}{M}\,\overline{V}$, where M is a double precision 3x3 matrix,
V is a double precision 3-vector.

☐ Other Library Modules:

Execution Time (microseconds): 304.4

Input Arguments:

Type	Precision	How Passed	Units
Matrix (3,3)	DP	R2 → 0th element	-
Vector (3)	DP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3.

Algorithm: Uses 2 nested loops, outer loop selecting rows of matrix, inner loop summing products of vector elements with current row elements.

MV6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MV6SN Size of Code Area: 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MV6SN

Function: Multiplies a single precision mxn matrix by a length n single precision vector.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overset{*}{M}\,\overline{V}$, where M is an mxn single precision matrix,
V is a length n single precision vector.

☐ Other Library Modules:

Execution Time (microseconds): 11.2 + m(11.0 + 18.4n)

Input Arguments:

Type	Precision	How Passed	Units
Matrix (m,n)	SP	R2 → 0th element	-
Vector (n)	SP	R3 → 0th element	-
Integer (m)	SP	R5	-
Integer (n)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Vector (m)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm: Exchange contents of R1 and R3 for more efficient addressing. Uses 2 nested loops, outer loops selecting rows of matrix, inner loop summing products of vector elements with current row elements.

MV6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MV6S3 Size of Code Area: 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MV6S3

Function: Multiplies a 3x3 single precision matrix by a length 3 single precision vector.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overset{*}{M}\,\overline{V}$, where M is a single precision 3x3 matrix,
V is a single precision 3-vector.

☐ Other Library Modules:

Execution Time (microseconds): 137.6

Input Arguments:

Type	Precision	How Passed	Units
Matrix (3,3)	SP	R2 → 0th element	-
Vector (3)	SP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,F0,F1,F2,F3.

Algorithm: Loops 3 times, each pass summing product of vector elements with current row elements and storing result in proper element output vector.

VM6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VM6DN Size of Code Area: 26 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VM6DN

Function: Multiplies length n double precision vector and nxm double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overline{V}\,\overset{*}{M}$ where V is a double precision n-vector, $n \neq 3$.
M is a double precision nxm matrix, $n \neq 3$.

☐ Other Library Modules:

Execution Time (microseconds): 23.2 + m(23.2 + 27.6n)

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	DP	R2 → 0th element	-
Matrix (n,m)	DP	R3 → 0th element	-
Integer(m)	SP	R6	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(m)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm: Uses 2 nested loops:
Outer loop selects matrix column.
Inner loop sums products of vector elements with matrix cloumn elements.

VM6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VM6D3 Size of Code Area: 24 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VM6D3

Function: Multiplies a length 3 double precision vector by a 3x3 double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overline{V} * M$, where V is a double precision 3-vector,
M is a double precision 3x3 matrix.

☐ Other Library Modules:

Execution Time (microseconds): 227.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-
Matrix(3,3)	DP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1,F2,F3,F4,F5.

Algorithm: Saves pointer to input vector (R2) so that R2 can be used to address both input vector and matrix by appropriate loading.
Loops 3 times:
Loads elements of vector into F0,F2,F4;
Switches R2 to point to matrix;
Sums products of column elements with vector elements;
Restore R2 to point at vector;
Make next pass.

VM6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VM6SN Size of Code Area: 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VM6SN

Function: Multiply a length n single precision vector by a nxm single precision matrix

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overline{V} * M$, where V is a single precision n-vector, $n \neq 3$,
M is a nxm matrix, $n \neq 3$.

☐ Other Library Modules:

Execution Time (microseconds): 12.4 + m(19.2 + 18.2n)

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2 → 0th element	-
Matrix(n,m)	SP	R3 → 0th element	-
Integer(m)	SP	R6	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(m)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm: Uses 2 nested loops; outer loop selecting matrix column, inner loop performs summation of products of vector elements and matrix column elements.

VM6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VM6S3 Size of Code Area: 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VM6S3

Function: Multiplies a length 3 single precision vector by a 3x3 single precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overline{V} * M$, where V is a single precision 3-vector,
M is a single precision 3x3 matrix.

☐ Other Library Modules:

Execution Time (microseconds): 141.2

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-
Matrix(3,3)	SP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→ 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1,F2,F3.

Algorithm: Uses one loop, looping three times, each pass addressing new column of matrix for explicit multiplication and summing, by elements of vector and storing into result. R1 is setup to contain both input matrix and output vector pointer in its two halves. Then circular shifts are used to place appropriate pointer into high HW for use as base.

VO6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VO6DN Size of Code Area: 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VO6DN

Function: Performs vector outer product of two double precision vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1 V2, where V1 and V2 are double precision n-vectors, n ≠ 3.

☐ Other Library Modules:

Execution Time (microseconds): 12.8 + n(5.8 + 24.4m)

Input Arguments:

Type	Precision	How Passed	Units
vector(n)	DP	R2 → 0th element	-
vector(m)	DP	R3 → 0th element	-
integer(n)	SP	R5	-
integer(m)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F4,F5.

Algorithm: Uses two loops on size of matrix(n):
Inner loop multiplies element of V1 by each element of V2 creating a row of result matrix.
Outer loop moves to next element of V1 and moves pointer to next row of result matrix.

VO6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VO6D3 Size of Code Area: 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VO6D3

Function: Computes vector outer product of length 3 double precision vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1 V2, where V1 and V2 are double precision 3-vectors.

☐ Other Library Modules:

Execution Time (microseconds): 251.0

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-
Vector(3)	DP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix (3,3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,F0,F1.

Algorithm: Same algorithm as VO6DN except that loop extents are set to literally 3.

VO6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VO6SN Size of Code Area: 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VO6SN

Function: Calculates vector outer product of 2 single precision vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1 V2, where V1 and V2 are single precision n-vectors, n ≠ 3.

☐ Other Library Modules:

Execution Time (microseconds): 14.2 + n(5.8 + 14.4m)

Input Arguments:

Type	Precision	How Passed	Units
vector(n)	SP	R2 → 0th element	-
vector(m)	SP	R3 → 0th element	-
integer(n)	SP	R5	-
integer(m)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F4,F5.

Algorithm: Same as VO6DN.

VO6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VO6S3 Size of Code Area: 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[xx] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VO6S3

Function: Calculates vector outer produce of 2 single precision length 3 vectors.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:
V1 V2, where V1 and V2 are single precision 3-vectors.

[ ] Other Library Modules:

Execution Time (microseconds): 160.6

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-
Vector(3)	SP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,F0,F1.

Algorithm: Same algorithm as VO6DN except that loop extents are set to literally 3.

VV0DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV0DN Size of Code Area: 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV0DN

Function:
Generates a double precision vector of length n, all of whose elements are the same.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:

$\overline{V} = 0$, where V is a double precision vector.

[x] Other Library Modules:
VV10D3

Execution Time (microseconds): 7.0 + 5.1n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R4,R5,F0,F1.

Algorithm: Uses loop counting down length (n); Stores input scalar into one element of vector on each pass through loop.

VV0DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV0DNP Size of Code Area: 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV0DNP

Function: Fills a column of a double precision matrix with a double precision scalar.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{n,*} = 0$; where M is a double precision matrix .

[ ] Other Library Modules:

Execution Time (microseconds): 7.0 + 7.2n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Integer(outdel)	SP	R7	-
Integer(length)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector (length)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R4,R5,R7,F0,F1.

Algorithm: Loops 'length' times;
Each pass through loop stores input scalar into vector element pointed to by R1 and then increments R1 by outdel.

VV0SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV0SN Size of Code Area: 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV0SN

Function: Generates a vector of length n, all of whose elements are the same.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:

$\overline{V} = 0$, where V is a single precision vector.

[X] Other Library Modules:
VV10S3

Execution Time (microseconds): 7.0 + 5.6n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Integer(n)	SP	R5	0

Output Results:

Type	Precision	How Passed	Units
Vector (n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R4,R5,F0,F1.

Algorithm: Uses loop counting down length of vector (n); Stores input scalar into one element of vector on each pass through loop.

VV0SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV0SNP Size of Code Area: 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV0SNP

Function: Moves a single precision scalar to all elements of a column of a single precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$M_{n,*} = 0$; where M is a single precision matrix .

☐ Other Library Modules:

Execution Time (microseconds): 7.0 + 6.0n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Integer(outdel)	SP	R7	-
Integer(length)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(length)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R4,R5,R7,F0,F1.

Algorithm: Loops 'length' times;
Each pass through loop stores input scalar into vector element pointed to by R1 and then increments R1 by outdel.

VV1DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1DN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced:

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1DN

Function: Moves a length-n double precision vector.
(Also used to move matrices).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overline{X} = \overline{Y}$, where X is a length-n double precision vector,
Y is a length-n double precision vector.

☐ Other Library Modules:

Execution Time (microseconds): 4.2 + 10.2n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1 → 0th argument	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm:

Loop n times; using indexing, BCTB on length; load and store each element, last element first.

VV1D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1D3 Size of Code Area: 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: _____

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1D3

Function: Moves a double precision 3-vector.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
$\overline{X} = \overline{Y}$, where X and Y are double precision 3-vectors

[ ] Other Library Modules:

Execution Time (microseconds): 25.2

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Load, then store each element.

VV1D3P

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1D3P Size of Code Area: 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1D3P

Function: Moves a length 3 double precision vector or row or column of a matrix to a vector or row or column of a matrix.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
$\overline{V} = \overline{M}_{1,*}$, where M is a 3x3 matrix and V is a 3-vector.

[ ] Other Library Modules:

Execution Time (microseconds): 46.0 if neither input nor output is contiguous.
48.4 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-
Integer (Indel)	SP	R6	-
Integer (Outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Performs single setup of size and then uses VV1DNP.
Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm: Initialize R5 with literal 3; Fall into VV1DNP routine;
R6, R7 specify distance in HW between input and output vector elements, respectively.

VV1D3P

Secondary Entry Name: VV1DNP

Function: Moves length n double precision vector or row or column of a matrix to a row or column vector.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: $\overline{V} = \overline{M}_{1,*}$, where M is an nxm matrix, V is an n-vector.

☐ Other library modules:

Execution Time (microseconds): 11.4n + 10.2 if neither input nor output is contiguous 11.4n + 12.6 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	DP	R2 → 0th element	-
Integer (Indel)	SP	R6	-
Integer (Outdel)	SP	R7	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector (n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm: Tests outdel, if 0, sets it to 4(HW); Tests indel, if 0, sets it to 4(HW); Loops 'length' times, adding indel to input pointer and outdel to output pointer each time. Each loop moves current input element to current output element.

VV1SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1SN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1SN

Function: Moves a length-n single precision vector. (Also used to move matrices).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: $\overline{V1} = \overline{V2}$, where V1 and V2 are single precision vectors of length n.

☐ Other Library Modules:

Execution Time (microseconds): 4.2 + 7.8n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	SP	R2 → 0th element	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector (n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm: Loop n times using indexing and BCTB on length. Load, then store each element, last element first.

VV1S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1S3 Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1S3

Function: Vector move, length 3, single precision.

Invoked by:

[ ] Compiler emitted code for HAL/S construct of the form:
$\overline{V1} = \overline{V2}$, where V1 and V2 are single precision 3-vectors.

[ ] Other Library Modules:

Execution Time (microseconds): 16.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Simple Load-Store sequence for each element.

VV1S3P

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1S3P Size of Code Area: 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1S3P

Function: Moves an SP 3-vector (or row or column of a matrix) to a 3-vector (or row or column of a matrix) when elements are not contiguous.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
$V = M_{1,*}$, where M is a 3x3 matrix, and V is a 3-vector.

[ ] Other Library Modules:

Execution Time (microseconds): 38.4 if neither input nor output is contiguous.
40.8 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-
Integer(indel)	SP	R6	Hw
Integer(outdel)	SP	R7	Hw

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
None.		

Comments: Performs simple setup of size for use by VV1SNP code.
Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm: Initialize R5 with literal 3;
Fall into VV1SNP routine;
R6, R7 specify distance in HW between input and output vector elements respectively.

VV1S3P

Secondary Entry Name: VV1SNP

Function:
Moves a length-n single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$V = M_{1,*}$, where M is an n x m matrix, and V is an n-vector.

☐ Other library modules:

Execution Time (microseconds): 8.6n + 10.2 if neither input nor output is contiguous. 8.6n + 12.6 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2 → 0th element	-
Integer(indel)	SP	R6	Hw
Integer(outdel)	SP	R7	Hw
Integer(length)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm: Tests outdel, if 0, sets it to 2 (halfwords).
Tests indel, if 0, sets it to 2 (halfwords).
Loops 'length' times, adding indel to input pointer and outdel to output pointer each time. Each loop moves current input element to current output element.

VV1TN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1TN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced:

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1TN

Function: Moves a length n double precision vector and converts it to single precision. (Also used to move matrices).

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
$\overline{V1} = \overline{V2}$, where V2 is a length n double precision vector and V1 is a single precision vector of declared length n.
☐ Other Library Modules:

Execution Time (microseconds): 4.2 + 9.0n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm: Using indexing and BCTB on length, loops, loading long and storing short, last element first.

VV1T3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1T3 Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: _____

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1T3

Function: Moves a length 3 double precision vector and converts it to single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\bar{V}1 = \bar{V}2$, where V1 is a single precision 3-vector, and V2 is a double precision 3-vector.

☐ Other Library Modules:

Execution Time (microseconds): 21.2

Input Arguments:

Type	Precision	How Passed	Units
Vector (3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector (3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Simple Load/Store for each element.

VV1T3P

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1T3P Size of Code Area: 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1T3P

Function: Moves a length 3 double precision vector or row or column of a matrix to a single precision vector or row or column of a matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$V=M_{1,*}$ Where V is a single precision 3-vector and M is a double precision 3x3 matrix.

☐ Other Library Modules:

Execution Time (microseconds): 38.4 if neither input nor output is contiguous.
40.8 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 →0th element	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 →0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm: Loads R5 with literal 3,
Falls into VV1TNP routine.

VV1T3P

Secondary Entry Name: VV1TNP

Function: Moves a double precision length or row column vector to a single precision length or row or column vector.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$V=M_{1,*}$ Where V is a declared lenth n single precision vector and M is a double precision nxm matrix.

☐ Other library modules:

Execution Time (microseconds): 8.6n + 10.2 if neither input nor output is contiguous. 8.6n + 12.6 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	DP	R2 → 0th element	-
Integer (outdel)	SP	R7	-
Integer (indel	SP	R6	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector (n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm: If outdel=0, set to 2 (HW);
If indel=0, set to 4(HW);
Loops 'length' times, adding indel to input pointer and outdel to output pointer each time. Each loop moves current input element to current output element.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

VV1WN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1WN **Size of Code Area:** 10 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1WN

Function: Moves a length-n single precision vector and converts it to double precision. (Also used to move matrices).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{X} = \overline{Y}$, where Y is a single precision length n vector, X is a double precision length n vector.

☐ Other Library Modules:

Execution Time (microseconds): 8.4 + 9.0n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	SP	R2 → 0th element	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm: Clear F0, F1. Loop using indexing on BCTB, last element first. Load short element into F0, Store long F0/F1 element.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

VV1W3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1W3 Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1W3

Function: Moves a length 3 single precision vector and converts it to double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overline{X} = \overline{Y}$, where Y is a single precision 3-vector, and
X is a double precision 3-vector.

☐ Other Library Modules:

Execution Time (microseconds): 23.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1.

Algorithm:

Clears F1;
Then explicit code to load (SP) each element of input vector and store (DP) into each element of result vector.

VV1W3P

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1W3P Size of Code Area: 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV1W3P

Function: Moves a single precision length 3 vector or row or column of a matrix, to a double precision vector or row or column of a matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overline{V} = \overline{M}_{1,*}$, where V is a double precision 3-vector, M is a single precision 3x3 matrix.

☐ Other Library Modules:

Execution Time (microseconds): 44.8 if neither input nor output is contiguous.
47.2 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector (3)	SP	R2 → 0th element	-
Integer (indel)	SP	R6	-
Integer (outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Sets up length for use by VV1WNP.
Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm: Loads R5 with literal 3, falls into VV1SNP.

Secondary Entry Name: VV1WNP

Function: Moves a length n single precision row or column vector to a double precision column vector.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: $V = M_{1,*}$, where V is declared as a length n double precision vector, and M is an nxm single precision matrix.

☐ Other library modules:

Execution Time (microseconds): 10.2n + 15.0 if either input or output is contiguous. 10.2n + 12.6 if neither input nor output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2 → 0th element	-
Integer (outdel)	SP	R7	-
Integer (indel)	SP	R6	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm: Clears F1. If outdel = 0, set to 4(HW); if indel = 0, set to 2(HW);
Loop 'length' times, adding indel to input pointer and outdel to output pointer each time. Each loop moves current input element to current output element.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV2DN **Size of Code Area:** 14 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV2DN

Function: Add two double precision vectors of lenth n.
(Also used to add two matrices.)

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: V1+V2, where V1 and V2 are double precision vectors of length ≠ 3.

☐ Other Library Modules:

Execution Time (microseconds): 8.8 + 20.6n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2 → 0th element	-
Vector(n)	DP	R3 → 0th element	-
Integer(n)	SP	R5	

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm: Uses indexing in load, add, store sequence controlled by BCTB on length.
Loading of an element is done with two LE instructions instead of on LED due to addressing inadequacies of R3 which is the input pointer.

VV2D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV2D3 Size of Code Area: 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV2D3

Function: Adds two double precision 3-vectors.

Invoked by:

[XX] Compiler emitted code for HAL/S construct of the form:
V1 + V2, where V1 and V2 are double precision 3-vectors.

[ ] Other Library Modules:

Execution Time (microseconds): 51.4

Input Arguments:

Type	Precision	How Passed	Units
Vector (3)	DP	R2 →0th element	-
Vector (3)	DP	R3 →0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector (3)	DP	R1 →0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Loads F0,F2,F4 with first half of each elemnt of V2 } Due to addressing
Loads F1,F3,F5 with second half of each elemnt of V2 } peculiarities of R3
Adds double from V1 to F0, F2, F4;
Stores double into elements of result.

VV2SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV2SN Size of Code Area: 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV2SN

Function: Add two single precision vectors of lenth n .
(Also used to add two matrices).

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
V1 + V2,where V1 and V2 are single precision vectors of lenth ≠ 3.

[ ] Other Library Modules:

Execution Time (microseconds): 8.4 + 13.6n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	SP	R2 →0th element	-
Vector (n)	SP	R3 →0th element	-
Integer (n)	SP	R5	

Output Results:

Type	Precision	How Passed	Units
Vector (n)	SP	R1 →0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm: Uses indexing in load, Add, Store sequence controlled by BCTB on lenth.

VV2S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV2S3 Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV2S3

Function: Add two single precision 3-vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1 + V2; where V1,V2 are single precision 3-vectors.

☐ Other Library Modules:

Execution Time (microseconds): 29.6

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-
Vector(3)	SP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Loads elements of V1 into F0,F2,F4.
Adds elements of V2 respectively.
Stores F0,F2,F4 into elements of result.

VV3DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV3DN Size of Code Area: 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV3DN

Function: Subtracts one double precision length n-vector from another.
(Also used to subtract matrices).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1 - V2, where V1 and V2 are double precision vectors of length ≠ 3.

☐ Other Library Modules:

Execution Time (microseconds): 6.0 + 22.7n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n) V1	DP	R2 → 0th element	-
Vector (n) V2	DP	R3 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm: Exchange contents of R2/R3 for addressing considerations.
Uses indexed load, subtract, store sequence controlled by BCTB on length. Load of minuend elements is done with two LE instructions due to use of R3 as index.

VV3D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV3D3 Size of Code Area: 24 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV3D3

Function: Subtracts two double precision vectors of length 3

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
V1 - V2 where V1 and V2 are double precision 3-vector

[ ] Other Library Modules:

Execution Time (microseconds): 55.4

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)V1	DP	R2 → 0th element	-
Vector(3)V2	DP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector (3)	DP	R1 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1.

Algorithm: Exchange contents of R2 and R3 for addressing considerations.
Load minuend elements into F0/F1, F2/F3, F4/F5 using two LE instruction each because of R3 addressing rules.
Subtract subtrahend elements
Store results using STED into result location.

VV3SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV3SN Size of Code Area: 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV3SN

Function: Subtracts one length n single precision vector from another.
(Also used to subtract matrices).

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
V2 - V1, where V1 and V2 are single precision vectors of length ≠ 3.

[ ] Other Library Modules:

Execution Time (microseconds): 8.4 + 13.6n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)V2	SP	R2 → 0th element	-
Vector(n)V1	SP	R3 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm: Uses indexed load, subtract, store sequence controlled by a BCTB loop on 'length'.

VV3S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV3S3 Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV3S3

Function: Subtracts two single precision vectors of length 3

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
V1-V2 whre V1 and V2 are single precision 3-vectors.

[ ] Other Library Modules:

Execution Time (microseconds): 29.6

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)V1	SP	R2 → 0th element	-
Vector(3)V2	SP	R3 → 0th elemnet	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Load minuend elements into F0,F2,F4
Subtract subtrahend elements from F0,F2,F4 respectively
Store F0,F2,F4 into result elements.

VV4DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV4DN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV4DN

Function: Multiplies each element of a double precision length n vector by a double precision scalar.
(Also used to multiply matrix by scalar).

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
$\overline{V}$ S, where V is a double precision vector of length ≠ 3, and S is a double precision scalar.

[ ] Other Library Modules:

Execution Time (microseconds): 7.0 + 23.4n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	DP	R2 → 0th element	-
Scalar	DP	F0	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector (n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm: Uses BCTB loop to count down 'length', performing load, multiply, store for each element.

VV4D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV4D3 Size of Code Area: 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV4D3

Function: Multiplies each element of a double precision vector of length 3 by a double precision scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V}$ S, where V is a length 3 double precision vector, and S is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 68.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Vector(3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3.

Algorithm: Simple load, multiply, store sequence for each element.

VV4SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV4SN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV4SN

Function: Multiplies a length-n single precision vector by a single precision scalar. (Also used to multiply matrix by scalar).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V}$ S, where V is a single precision vector of length ≠ 3, and S is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 7.0 + 14.0n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Vector(n)	SP	R2 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm: Uses BCTB loop to count down 'length', performing load, multiply, store for each element.

VV4S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV4S3 Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV4S3

Function: Multiplies each element of a single precision 3-vector by a single precision scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V}$ S, where V is a single precision 3-vector, and
S is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 38.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Vector(3)	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3.

Algorithm: Simple load, multiply, store for each element.

VV5DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV5DN Size of Code Area: 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV5DN

Function: Divides a double precision vector of length n by a double precision scalar. (Also used to divide matrix by scalar).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V}$/S, where V is a double precision vector of length ≠ 3, and
S is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 37.0 + 24.2n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Vector (n)	DP	R2 → 0th element	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector (n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
25	Scalar argument is zero.	Store original vector as result.

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm: Test F0; if zero, preset quotient to 1; otherwise, compute 1/S and then use BCTB loop to count down 'length' performing load, multiply (by 1/S), store sequence for each element.

VV5D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV5D3 Size of Code Area: 26 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV5D3

Function: Divide each element of a double precision length 3 vector by a double precision scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V}/S$, where V is a double precision 3-vector, and
S is a double precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 98.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Vector(3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
25	Scalar argument is zero.	Store original vector as result.

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Test F0; if zero, send error and set quotient to 1;
Otherwise, quotient 1/arg is calculated and then used in simple
Load, multiply, store sequence for each element.

VV5SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV5SN Size of Code Area: 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV5SN

Function:
Divides single precision vector of length n by single precision scalar.
(Also used to divide matrix by scalar).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V}/S$, where V is a single precision vector of length ≠ 3, and
S is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 7.2 + 18.0n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Vector (n)	SP	R2 → 0th element	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
25	Scalar argument is zero.	Store original vector as result.

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm: Test F0, if zero, set F0 to 1; Uses BCTB loop to count down
'length' performing. Load, divide, store sequences for each element.

VV5S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV5S3 **Size of Code Area:** 18 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 2 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV5S3

Function: Divide each element of a single precision vector of length 3 by a single precision scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V}/S$, where V is a single precision 3-vector, and
S is a single precision scalar.

☐ Other Library Modules:

Execution Time (microseconds): 50.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Vector(3)	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
25	Scalar argument is zero.	Store original vector as result.

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3.

Algorithm: Test F0; if zero, set F0 to floating point 1; then simple load, divide, store sequence for each element.

VV6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV6DN **Size of Code Area:** 12 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV6DN

Function: Forms dot product of two double precision length n vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

V1.V2, where V1 and V2 are double precision vectors of length n, $n \neq 3$.

☐ Other Library Modules:

Execution Time (microseconds): 16.4 + 25.4n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	DP	R2 → 0th element	-
Vector (n)	DP	R3 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1,F2,F3.

Algorithm: Loads R3 into R1 for addressability advantages performs:

$$\sum_{i=1}^{n} V1_i \, V2_i$$

by loops counting down n;
Each pass loads $V1_i$ multiplies by $V2_i$ and add to accumulated sum in F0.

VV6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV6D3 Size of Code Area: 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV6D3

Function: Forms dot product of 2 double precision 3-vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1.V2 where V1 and V2 are double precision 3-vectors.

☐ Other Library Modules:

Execution Time (microseconds): 71.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-
Vector(3)	DP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R2,R3,R4,F0,F1,F2,F3.

Algorithm: Moves R3 to R1 for addressability advantages.
Performs:

$$\sum_{i=1}^{n} V1_i \, V2_i$$

via straight line code, no loops, accummulating result in F0 .

VV6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV6SN Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV6SN

Function: Forms dot product of two length n single precision vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1.V2 where V1 and V2 are single precision n-vectors, $n \neq 3$.

☐ Other Library Modules:

Execution Time (microseconds): 15.2 + 16.8n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	SP	R2 → 0th element	-
Vector (n)	SP	R3 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,F5,F0,F1,F2,F3.

Algorithm: Move R3 to R1 for addressability advantages
performs:

$$\sum_{i=1}^{n} V1_i \, V2_i$$

by a loop counting down n ;
Each pass loads $V1_i$, multiplies by $V2_i$ and adds to accumulated sum in F0.

VV6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV6S3 **Size of Code Area:** 10 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV6S3

Function: Forms dot product of two single precision 3-vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1.V2 where V1 and V2 are single precision 3-vectors.

☐ Other Library Modules:

Execution Time (microseconds): 41.8

Input Arguments:

Type	Precision	How Passed	Units
Vector (3)	SP	R2 → 0th element	-
Vector (3)	SP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	FO	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R2,R3,R4,F0,F1,F2,F3.

Algorithm: Calculates $V1_1 V2_1 + V1_2 V2_2 + V1_3 V2_3$ via direct code, no loops, accummulating result in FO.

VV7DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV7DN **Size of Code Area:** 8 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV7DN

Function: Vector negate, double precision, length n.
(Also used to negate matrices).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$-\overline{V}$, where V is a double precision vector of length n, n≠3.

☐ Other Library Modules:

Execution Time (microseconds): 7.0 + 11.4n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	DP	R2 → 0th element	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm: Uses loop to count down 'n', each pass performing load, negate, store sequence on current vector element.

VV7D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV7D3 Size of Code Area: 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV7D3

Function: Vector negate, double precision for vectors of length 3.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$-\overline{V}$, where V is a double precision 3-vector.

☐ Other Library Modules:

Execution Time (microseconds): 32.4

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Simple, direct code sequence, no loops.
Performs 3 loads, 3 negates, 3 stores.

VV7SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV7SN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV7SN

Function:
Vector negate, single precision, length n.
(Also used to negate matrices).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$-\overline{V}$, where V is a single precision vector of length n, n≠3.

☐ Other Library Modules:

Execution Time (microseconds): 7.0 + 9.0n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	SP	R2 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm: Uses loop to count down 'n', each pass performing load, negate, store sequence on current vector element.

VV7S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV7S3 Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV7S3

Function:

Vector negate, single precision for vectors of length 3.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$-\overline{V}$, where V is a single precision 3-vector.

☐ Other Library Modules:

Execution Time (microseconds): 23.4

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector (3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Direct, inline code, no loops.
Does 3 loads, 3 negates, 3 stores.

VV8D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV8D3 Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV8D3

Function: Compares two double precision 3-vectors

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
IF $\overline{X} = \overline{Y}$... , where X and Y are double precision 3-vectors.

☐ Other Library Modules:

Execution Time (microseconds): 59.0 if X=Y; 16.2n + 24.6 if X≠Y where n = 3 -(index of last non-matching pair of elements).

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-
Vector (3)	DP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
equal/not equal	-	condition code	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm: Loads a literal 3 into R5, then drops into VV8DN code.

VV8D3

Secondary Entry Name: VV8DN

Function: Compares two double precision vectors of length n.
(Also used to compare matrices).

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**
IF $\overline{X} = \overline{Y}$, where X and Y are double precision vectors of length n, n≠3.

☐ **Other library modules:**

Execution Time (microseconds): 16.2n + 18.0 if X=Y; 16.2m + 22.2 if X≠Y, where m = n-(index of last non-matching pair of elements)

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	DP	R2 → 0th element	-
Vector (n)	DP	R3 → 0th element	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
equal/unequal	-	Condition code.	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm: Loads R3 into R1 for better addressability. Loops, counting down 'size', each pass compares values of one element of each vector. When first non-compare occurs, branch to return point is taken, exiting loop.

Condition code is set based upon whether count down loop reaches 0. Condition code of 00 indicates equality, 01 indicates unequality.

VV8S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV8S3 Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV8S3

Function: Compares two single precision vectors of length 3.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
IF $\overline{X} = \overline{Y}$..., where X and Y are single precision 3-vectors.

☐ Other Library Modules:

Execution Time (microseconds): 42.8 if X=Y; 10.8n + 8.4 if X≠Y, where n = 4 -(index of last non-matching pair of elements).

Input Arguments:

Type	Precision	How Passed	Units
Vector (3)	SP	R2 → 0th element	-
Vector (3)	SP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
equal/not equal	-	Condition Code.	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm: Loads a literal 3 into R5. Then falls into VV8SN routine.

Secondary Entry Name: VV8SN

Function: Compares two single precision vectors of length n. (Also used to compare matrices).

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:** IF X = Y... , where X and Y are single precision vectors of length n, n≠3.

☐ **Other library modules:**

Execution Time (microseconds): 10.8n + 8.0 if X=Y; 10.8m + 6.0 if X≠Y, where m = n -(index of last non-matching pair of elements) +1.

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	SP	R2 → 0th element	-
Vector (n)	SP	R3 → 0th element	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
equal/not equal	-	Condition Code.	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm: Loads R3 into R1 for better addressability. Loops counting down 'size', each pass compares values of one element of each vector. When first non-compare occurs, branch to return point is taken, exiting loop. Condition code is set based upon whether count down loop reaches 0. Condition code of 00 indicates equality, 01 indicates inequality.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV10D3 **Size of Code Area:** 56 Hw

Stack Requirement: 20 Hw **Data CSECT Size:** 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: DSQRT , VV0DN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV10D3

Function: Creates unit vector of length 3 for input 3-vector in double precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

UNIT($\bar{V}$), where V is a double precision 3-vector.

☐ Other Library Modules:

Execution Time (microseconds): 402.7

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R4 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
28	Input vector has all elements=0.	Return input vector.

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Loads R5 with literal 3, then drops into VV10DN code.

Secondary Entry Name: VV10DN

Function:
Creates unit vector of length n for input vector of length n in double precision.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**
UNIT($\overline{V}$), where V is a double precision vector of length n, n≠3.

☐ **Other library modules:**

Execution Time (microseconds): 259.7 + 47.8n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	DP	R4 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
28	every element of input vector is 0.	Return input Vector

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Uses loop to sum squares of elements of input vector. Calls DSQRT to get square root of sum. Uses loop to divide each element of input vector by square root value and store into result vector.

Secondary Entry Name: VV9D3

Function: Calculates magnitude of length 3 double precision vector.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**
ABVAL($\overline{V}$), where V is a double precision 3-vector.

☐ **Other library modules:**

Execution Time (microseconds): 300.2

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Loads R5 with literal 3, then drops into VV9DN code.

Secondary Entry Name: VV9DN

Function:
Calculates magnitude of length n double precision vector.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
ABVAL($\overline{V}$), where V is a double precision vector of length n.

☐ Other library modules:

Execution Time (microseconds): 226.6 + 24.4n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2 → 0th element.	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Uses loop counting down size (n), each pass squaring an element of input vector and adding to accumulated value in F0; after loop, calls DSQRT to obtain final result in F0.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV10S3 **Size of Code Area:** 50 Hw

Stack Requirement: 24 Hw **Data CSECT Size:** 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: SQRT, VV0SN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VV10S3

Function: Creates unit vector of length 3 for input 3-vector in single precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
UNIT($\overline{V}$), where V is a single precision 3-vector.

☐ Other Library Modules:

Execution Time (microseconds): 236.4

Input Arguments:

Type	Precision	How Passed	Units
Vector (3)	SP	R4 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector (3)	SP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
28	Input vector has all elements = 0.	Return input vector.

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Loads R5 with literal 3, then drops into VV10SN code.

VV10S3

Secondary Entry Name: VV10SN

Function: Creates unit vector of length n for input vector of length n in single precision.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**
UNIT($\overline{V}$), where V is a single precision vector of length n, n≠3.

☐ **Other library modules:**

Execution Time (microseconds): 130.6 + 32.8n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R4 → 0th element	-
Integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R2 → 0th element	-

Errors Detected:

Error #	Cause	Fixup
28	Sum of squares of all elements = 0.	Return zero vector.

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:
Uses loop to sum squares of elements of input vector.
Calls SQRT to get square root of sum.
Uses loop to divide each element of input vector by square root return value and store into result vector.

VV10S3

Secondary Entry Name: VV9SN

Function: Calculates magnitude of single precision vector of length n.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**
ABVAL($\overline{V}$), where V is a single precision vector of length n, n≠3.

☐ **Other library modules:**

Execution Time (microseconds): 118.9 + 14.0n

Input Arguments:

Type	Precision	How Passed	Units
Vector (n)	SP	R2 → 0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:
Uses loop counting down size (n), each pass squaring an element of input vector and adding to accumulated value in F0; After loop, calls SQRT to obtain final result in F0.

Secondary Entry Name: VV9S3

Function:
Calculates magnitude of length 3 single precision vector.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**
ABVAL($\overline{V}$), where V is a single precision 3-vector.

☐ **Other library modules:**

Execution Time (microseconds): 168.3

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Loads R5 with literal 3; falls into VV9SN code.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VX6D3 **Size of Code Area:** 36 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VX6D3

Function: Forms cross product of 2 double precision 3-vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$\overline{X} * \overline{Y}$, where X and Y are double precision vectors of length 3.

☐ Other Library Modules:

Execution Time (microseconds): 137.6

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2 → 0th element	-
Vector(3)	DP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1,F2,F3,F4,F5.

Algorithm: Direct code, no loops, to calculate cross product.

VX6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VX6S3 Size of Code Area: 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VX6S3

Function: Performs vector cross product of two single precision length 3 vectors.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\vec{X} * \vec{Y}$ where X and Y are single precision 3-vectors.

☐ Other Library Modules:

Execution Time (microseconds): 78.0

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2 → 0th element	-
Vector (3)	SP	R3 → 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1 → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1,F2,F3.

Algorithm: Uses direct code, no loops, to calculate.
$(X_2 Y_3 - X_3 Y_2, X_3 Y_1 - X_1 Y_3, X_1 Y_2 - X_2 Y_1)$

5.3.4 Character Routine Descriptions

This subsection presents those routines which manipulate character data. Routines which convert to and from character data are not included here. Such routines are found under Section 5.3.6.

CASPV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CASPV Size of Code Area: 64 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CASPV

Function: Assigns a partition of a character string to a temporary.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

... C$(I TO J) ... where C is a character string.

☒ Other Library Modules: CPASP

Execution Time (microseconds): (See next page)

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2 → descriptor	-
Integer (I)	SP	R5	-
Integer (J)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Character (temporary)	-	R1 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Indices out-of-bounds for input string.	Set out-of-bounds index to first or last character of string.

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6.

Algorithm: Several checks for possible errors are made before the transfer of characters is actually done. The index to the first character (I) is checked to be not less than 1. If it is, then set to 1. The index to the last character (J) is checked to be less than the length of the source string. If it is greater, then the index is set equal to the current length. Third, if J < I, the fixup is the NULL string. If the input actually is the NULL string, then no error is signalled. Finally, if the partition length exceeds the max length of the destination string, then the partition is truncated.

(Continued on next page)

CASPV

Execution Time:

if p = 0 43.8

if p > 0:

52.0 + 3.8 (if I is even)

+ 9.4k (if k is odd)

+ 13.1k (if I is even)

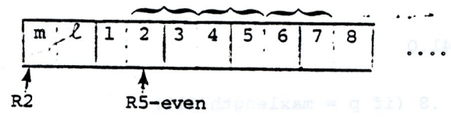
where p = minimum (J-I+1, 255)

k = ceiling (P/2)

Algorithm (Con't)

All that remains to be done is the halfword-by-halfword transfer. The character count is incremented by one before dividing by two so that the halfword count is rounded to the next highest halfword if the character count was odd.

If I (the first character index) is odd then the transfer is straightforward. If even, then there are alignment problems to work around. The odd byte of the first halfword to move must not be moved, so halfwords crossing the "natural" halfword boundary are moved instead.



Secondary Entry Name: CASP

Function: Assigns a partition of a character string to a receiver string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

C1 = $C2_{I\ TO\ J}$, where C1 and C2 are character variables, and I and J are integers.

☐ Other library modules:

Execution Time (microseconds): (See below)

Input Arguments:

Type	Precision	How Passed	Units
character (C2)	-	R2 → descriptor	-
Integer (I)	SP	R5	-
Integer (J)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
character(C1)	-	R1 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Index out-of-bounds for input string.	Set out-of-bounds index to first or last character of string.

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6.

Algorithm:

Same as CASPV, except destination is a variable instead of a temporary.

Execution Time: if p = 0: 41.0
if p > 0:

49.2 + .8 (if p = maxlength(C1))
+ 3.8 (if I is even)
+ 9.4k (if I is odd)
+ 13.1k (if I is even)

where p = minimum (J-I+1, maxlength(C1))

k = ceiling (P/2).

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CASV **Size of Code Area:** 28 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CASV

Function: Character assign for output; assigns string from data to I/O buffer area.

Invoked by:

☐ Compiler emitted code for HAL/S construct of the form:

☒ Other Library Modules: COUTP, CINP

Execution Time (microseconds): if C2 is null_string: 29.2
if C2 ≠ null string: 40.2 + 9.4 (ceiling (P/2-1)) + .8 (if length(C2) > maxlength(C1)), where p = minimum (length(C2), maxlength(C1)).

Input Arguments:

Type	Precision	How Passed	Units
character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
character	-	R1 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5.

Algorithm: First, the max length of the destination string is set to 255. Then, the length descriptor halfword of both the source string and the destination string are examined. The min of the max length of the destination and the current length of the source is taken as the new currlength of the destination. Next, the number of halfwords to move is found by incrementing the character count by one (in case the character cound is odd) and dividing by two. If the source is a null string, the routine exits. If the character count is odd, the last byte in the string is moved anyway since it is always ignored. The assignment is made by moving the string halfword-by-halfword to the location specified by the destination pointer.

CASV

Secondary Entry Name: CAS

Function: Character assignment, non-partitioned.

Invoked by:

☑ Compiler emitted code for HAL/S construct of the form:
C1 = C2, where C1 and C2 are character strings.

☒ Other library modules: CIN

Execution Time (microseconds): if input is null string: 32.0
if input ≠ null string: 43 + 9.4 · (ceiling(P/2-1)), where p = length of input character string.

Input Arguments:

Type	Precision	How Passed	Units
character (C2)	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
character (C2)	-	R1 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5.

Algorithm: Same as CASV, except MAXLEN of destination is not set to 255, but left with original MAXLEN value.

CATV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CATV Size of Code Area: 76 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☑ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CATV

Function: Catenates two character strings and stores into a temporary.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
X||Y

☐ Other Library Modules:

Execution Time (microseconds): Times depend on whether first source string = destination string and whether the first source string has an odd character count creating an alignment problem. (Continued on next page).

Input Arguments:

Type	Precision	How Passed	Units
Character (X)	-	R2 → descriptor	-
Character (Y)	-	R3 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character(temporary)	-	R1 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
none		

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm: The lengths of the source strings are checked against the destination string for legal values. The second source string may be truncated if its length + that of the first source string exceed the length of the destination. If the first source string and the destination string are the same string (found by comparing addresses), then only the second source string is moved. After checking these things, the routine needs only to actually move the strings. The first is a straight halfword-by-halfword move. If its length is odd, then there is the alignment problem to contend with. The second string is moved starting where the first one left off. See description of CASPV for what is done when the first source string has an odd character count.

CATV

Execution Time (microseconds): (Cont'd.)

if X is null string and Y is null: 52.2 μsec.

if X and Y are not both null: XTIME + YTIME.

XTIME: if X is null string: 24.0

if X ≠ null string: 29.8 + 9.4 (ceiling (P/2))

where p = length(X).

YTIME: if Y is null string: 27.8

if Y is ≠ null string:

52.1 + 14.1 · (ceiling(Q/2-1)) + 6.0 (if P+Q is odd) } if P is odd

32.3 + 9.4 · (ceiling (Q/2)) if P is even

where Q = minimum (length(Y), 255-P).

Secondary Entry Name: CAT

Function: Catenates two character strings and stores into a third string.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**
Not used yet.

☐ **Other library modules:**

Execution Time (microseconds): Same as CATV - 2.5

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2 → descriptor	-
Character	-	R3 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character	-	R1 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm: Same as CATV, except string is moved to real data area, not to a temporary.

CINDEX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CINDEX Size of Code Area: 52 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CINDEX

Function: Performs HAL/S INDEX function: finds occurrance of one character string within another.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
INDEX(A,B), where A and B are character strings; B is searched for within A.

☐ Other Library Modules:

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
character(A)	-	R2 → descriptor	-
character(B)	-	R4 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R5,F0,F1,F2,F3,F4,F5.

Algorithm:
1) If either string is null, return zero.
2) Set pointer to first character of A.
3) If size of B exceeds size of A, beyond A pointer, return zero.
4) Loop on size of B, comparing elements of A and B beginning at current A pointer; on non-equality go to step 6.
5) Comparison loop in 4 succeeded, return current A pointer.
6) Increment A pointer by one byte, go to step 3.

CINDEX

Execution Time (microseconds):

if A is null: 32.8

if B is null: 38.0

if length(B) > length(A): 44.8

if result = 0:

$$time = 38.0 + \sum_{I=1}^{JA} \left(\sum_{J=1}^{KA_I} (15.4 + KB_J + KB_{J+I-1}) + 16.4 + KB_I \right)$$

where JA = 2 (length(C1) - length(C2))+1

KA_I = # of compares required to determine that C1$(length(C2) at I) ¬= C2.

$$KB_X = \begin{cases} 14.4 \text{ if X is even} \\ 15.6 \text{ if X is odd.} \end{cases}$$

if result ¬= 0:

$$time = 29.6 + \sum_{I=1}^{result} \left(\sum_{J=1}^{KA_I} (15.4 + KB_J + KB_{J+I-1}) + 16.4 + KB_I \right) - KB_{result}$$

where KA_I = # of comparisons required to determine that C1$(length(C2) at I) ¬= C2 if I ¬= result.
length(C2) if I = result.

KB_X is as above.

CLJSTV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CLJSTV Size of Code Area: 40 Hw

Stack Requirement: - 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: GTBYTE, STBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CLJSTV

Function: Left justifies a character string to a specified length by
1) padding on the right with blanks if too short;
2) truncating on the right if too long.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
LJUST(A,B), where A is a character string, and
B is an integer.
☐ Other Library Modules:

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
character(A)	-	R4 → descriptor	-
integer (B)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
character(temporary)	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
18	Input string length greater than requested size. or B < 0.	Truncate input string to specified size.

Comments:
Registers Unsafe Across Call: F0,F1.

Algorithm: Compares requested length to 255 and retains smaller as L;
Compares L with input string length:
if greater, truncates on right to length L and moves to output;
if same, moves input string unchanged to output;
if less, pads on right with blanks and moves to output.

CLJSTV

Execution Time:

34.0 + 2.8 (if B < 255)
+ 2.0 (if n > 0)
+ 40.8n
+ 1.6 (if n is odd)
+ 0.4 (if m ≤ 0)
+ 1.0 (if m is odd and n is even)
- 1.0 (if m is odd and n is odd)
+ 23.8 m

where n = length(A)
m = maximum(B-n,0)

CPAS

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPAS Size of Code Area: 80 Hw

Stack Requirement: 20 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: GTBYTE, STBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CPAS

Function: Assigns a character string to a partition of another string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$C2_{I\ TO\ J} = C1$; C1 and C2 are character strings.

☒ Other Library Modules: CPASP, CINP

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
character string (source) (C1)	-	R4 → descriptor	-
integer (I)	-	R5	-
integer (J)	-	R6	-

Output Results:

Type	Precision	How Passed	Units
character (destination) (C2)	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Index out-of-bounds for destination string. or J < I-1	Set out-of-bounds index to first or last character of destination.

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm: First, the length of the partition is compared to the length of the source. If the source is longer, truncate it. If the destination partition is longer, then pad with blanks. The character count is determined and the string is moved byte-by-byte with the GTBYTE and STBYTE routines.

CPAS

Execution Time (microseconds):

$$34.2 + KA + \sum_{k=1}^{LHP} (5.6 + KC_{LOUT+K}) + KD$$

$$+ \sum_{k=1}^{NCHAR} (7.6 + KC_{I+K-1} + KF_k) + KE$$

$$+ \sum_{k=1}^{RHP} (5.6 + KC_{I+LIN+K-1}) + KG$$

where

LOUT = length(C2) before assignment

LIN = length(C1)

$$KA = \begin{cases} 25.4 & \text{if } J <= LOUT \\ 34.0 & \text{if } J > LOUT \end{cases}$$

LPART = J-I+1 (length of partition)

$$KB = \begin{cases} 9.2 & \text{if LPART > 0 and LPART <= LIN} \\ 13.8 & \text{if LPART > 0 and LPART > LIN} \\ 0 & \text{otherwise} \end{cases}$$

LHP = I-LOUT-1

$$KC_X = \begin{cases} 19.2 & \text{if X is odd} \\ 17.2 & \text{if X is even} \end{cases}$$

$$KD = \begin{cases} 4.0 & \text{if LHP <= 0} \\ 0 & \text{otherwise} \end{cases}$$

NCHAR = MINIMUM(LPART,LIN)

$$KE = \begin{cases} .8 & \text{if NCHAR = 0} \\ 0 & \text{otherwise} \end{cases}$$

Note that in summations, if start_index > end_index then summation goes to 0.

$$KF_X = \begin{cases} 15.6 & \text{if X is odd} \\ 14.4 & \text{if X is even} \end{cases}$$

RHP = LPART-LIN

$$KG = \begin{cases} .4 & \text{if RHP <= 0} \\ 0 & \text{otherwise} \end{cases}$$

CPASP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPASP Size of Code Area: 16 Hw

Stack Requirement: 146 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: CASPV, CPAS

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CPASP

Function: Assigns a partition of a character string into a partition of another character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$C1_{I \text{ TO } J} = C2_{K \text{ TO } L}$; C1 and C2 are character strings, and I,J,K,L are integers.

☐ Other Library Modules:

Execution Time (microseconds): 42.8 + time for CASPV and CPAS.

Input Arguments:

Type	Precision	How Passed	Units
character(source)C2	-	R4 → descriptor	-
integer(K)	SP	R5	-
integer(L)	SP	R6	-
integer(I||J)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
character(destination) C1		R1 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Subscript of character string out of bounds.	Set out-of-bounds value to first or last character of associated string.

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm: The input partition is put into a VAC by the CASPV routine. The index arguments of the destination string and pointers are set up for the CPAS routine, that then moves the contents of the VAC into the destination string.

CPR

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPR Size of Code Area: 46 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CPR

Function: Compares two character strings for '=' or '¬=' and sets condition code.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
IF C1 = C2..., where C1 and C2 are character strings.

☒ Other Library Modules: CPRA

Execution Time (microseconds): (see next page).

Input Arguments:

Type	Precision	How Passed	Units
Character (C1)	-	R2 → descriptor	-
Character (C2)	-	R3 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
equal/not equal	-	condition code	-

Errors Detected:

Error #	Cause	Fixup

Comments: In order to not change the condition code after the comparisons and before exiting, instructions that change the c.c. are replaced by those that do not change it. For example, LH replaced by IHL and SLL. (Cont'd. below)

Algorithm: See CPRC entry.

Comments: (Cont'd.)

Registers Unsafe Across Call: R2,R3,R4,R5,R6.

CPR

Execution Time (microseconds):

If C1 and C2 do not halfword compare and K $\geq$ 2:

setup + 11.6J + 12.9

If K is even or K = 0 and C1 and C2 halfword compare 1 up till the Kth character:

setup + 11.6n + 20.1

If K is odd and C1 = C2 up till the Kth character:

setup + 11.6n + 29.9

If K is odd and only the last characters compared differ

setup + 11.6n + 20.3

where:

K = minimum(length(C1), length(C2))

setup = 23 + :4 (if length(C2) < length(C1))

J = number of matching halfword compares

n = floor (K/2)

Secondary Entry Name: CPRC

Function: Compares two character strings for collating sequence and sets condition code.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**
If C1 < C2..., or any other relational operator, except '=', or '¬='.

☐ **Other library modules:**

Execution Time (microseconds): Same as CPR

Input Arguments:

Type	Precision	How Passed	Units
character (C1)	-	R2 - descripter	-
character (C2)	-	R3 - descripter	-

Output Results:

Type	Precision	How Passed	Units
relation	-	condition code	-

Errors Detected:

Error #	Cause	Fixup

Comments: See CPR

Registers Unsafe Across Call: R2,R3,R4,R5,R6.

Algorithm: Find the smaller of the lengths of the two strings to be compared. Compare this many characters halfword-by-halfword, and compare the upper bytes of the last halfwords separately if the character count is odd. If any of these comparisons are unequal, then return the resultant condition code. If all are equal, then compare the lengths of the two strings, and return the resultant code.

CPRA

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPRA Size of Code Area: 20 Hw

Stack Requirement: 22 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: CPR

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CPRA

Function: Compares arrays of character strings.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
IF S1 = S2, where S1 and S2 are structures, one of whose nodes is a length n array of character strings.

☐ Other Library Modules:

Execution Time (microseconds): $23.2 + \sum_{k=1}^{NCMP} (18.2 + CPRTIME_k) - 14.2$ (if arrays are not equal)
(Cont'd. below).

Input Arguments:

Type	Precision	How Passed	Units
character array	-	R2 → 0th element	-
character array	-	R3 → 0th element	-
integer(# Hw in each string)	SP	R6	-
integer(n)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
equal/not equal	-	condition code	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: None.

Algorithm: Pointers to character strings within the array are set, then CPR routine called. If all pairs of strings within the array are equal, result of CPRA is "equal", otherwise the result is "not equal".

Execution Time (Cont'd.)

where NCMP = number of elements in arrays if arrays are equal, index of first non-matching character strings in arrays if arrays not equal.

$CPRTIME_X$ = time in CPR for S1.C$(X:) and S2.C$(X:) where C is the node for the array of character strings.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

CRJSTV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CRJSTV Size of Code Area: 46 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: GTBYTE, STBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CRJSTV

Function: Right-adjusts a character string to a specified length by:
1) padding on left with blanks if too short;
2) truncating on left if too long.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
RJUST(A,B), where A is a character string, and B is an integer.

☐ Other Library Modules:

Execution Time (microseconds): (See next page).

Input Arguments.

Type	Precision	How Passed	Units
character (A)	-	R4 → descriptor	-
integer (B)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Character (temporary)	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
18	Input string length greater than input size or B < 0.	Truncate input string on left to proper size.

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm: Compares requested length to 255 and retain smaller as L;
Compares current length with L:
if greater, truncates on left and moves to input;
if same, moves string to output;
if less, pads on left and moves input string to output.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

CRJSTV

Execution Time (microseconds):

$$34.8 + KA + \sum_{k=1}^{NBLANK} (6.8 + KB_k) + KC$$

$$+ \sum_{k=1}^{NCHAR} (5.6 + KD_k + KB_{NBLANK+K}) + KE$$

where:

$$KA = \begin{cases} 0 & \text{if } B >= 255 \\ 2.8 & \text{if } B < 255 \end{cases}$$

$$NBLANK = \begin{cases} B\text{-length}(A) & \text{if } B > \text{length}(A) \\ 0 & \text{otherwise} \end{cases}$$

$$KB_X = \begin{cases} 19.2 & \text{if X is odd} \\ 17.2 & \text{if X is even} \end{cases}$$

$$KC = \begin{cases} 1.2 & \text{if NBLANK} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$NCHAR = \text{length}(A)$$

$$KD_X = \begin{cases} 15.6 & \text{if X is odd} \\ 14.4 & \text{if X is even} \end{cases}$$

$$KE = \begin{cases} .4 & \text{if NCHAR} = 0 \\ 0 & \text{otherwise} \end{cases}$$

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTRIMV Size of Code Area: 94 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: GTBYTE, STBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CTRIMV

Function: Implements HAL/S TRIM function - strips leading and trailing blanks from a character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
TRIM(C), where C is a character string.

☐ Other Library Modules:

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
character (C)	-	R4 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
character (temporary)	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1.

Algorithm: Because there are no character or byte compare instructions on the AP-101, the routine first tests length of string. If odd, it sets R7 to 1. "Length" is shifted right 1, resulting in length in # of halfwords (-1 if odd). Compares first halfword with bb, continues comparing consecutive halfwords of string with bb, until a halfword that is not equal to bb is found. Then tests this halfword to see if first byte is b. Adds length of string in halfwords to pointer to string, resulting in a pointer to end of string. Compares last halfword of string with bb. If equal, then moves pointer back a halfword and again compares. When a halfword not equal to bb is found, the halfword is tested to see if it is Cb or CC (where C stands for any character). Length of string is appropriately adjusted, and routine branches to a character move loop.

CTRIMV

Execution Time (microseconds):

If length(C) = 0: 30.4

If length(C) = 1 and C is a blank: 64.0

If length(C) = 1 and C is not a blank: 102.8

If length(C) > 1 and all blank

$$44.6 + KA + 13.2KB$$

where:

KA = 0 if length(C) is even
19.4 if length(C) is odd

KB = floor (length(C)/2)

If length(C) > 1 and not all blank

$$60.4 + KA + 13.2 \cdot KB + KC + KD(11.6 \cdot KE + 13.6 + KF) + \sum_{k=1}^{NCHAR} (39.2 + KG_{KH+k-1} + KI_k)$$

where:

KA = 0 if length(C) is even
19.4 if length(C) is odd and C$(#) is blank
18.4 if length(C) is odd and C$(#)¬= blank
and C$(#)¬= null
22.4 if length(C) is odd and C$(#)¬= blank
and C$(#) = null

KB = # halfwords = blank||blank at the beginning of C

KC = 0 if index of first non_blank character is odd
and this character = null
4.8 if index of first non blank character is odd
and this character = null
9.0 if index of first non blank character is even.

KD = 0 if length(C) is odd and (C$(#)¬= blank
1 otherwise.

KE = # halfwords = blank||blank at the end of C

(Continued on next page)

CTRIMV

Execution Time (Continued):

KF = 0 if index of last untrimmed character is even and
this character ¬= null
3.6 if index of last untrimmed character is even and
this character = null
4.4 if index of last untrimmed character is odd.

NCHAR = length of result.

KG_X = 0 if X is even
2.0 if X is odd

KH = index of first non blank character

KI_X = 0 if X is odd
1.2 if X is even

5.3.5 Array Function Routine Descriptions

This subsection presents those routines which are classed as "ARRAY FUNCTIONS" by the HAL/S Language Specification. These are routines which operate upon arrayed arguments and produce a single element result.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DMAX Size of Code Area: 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DMAX

Function: Finds maximum value in a double precision scalar array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

MAX(<DP array>)

☐ Other Library Modules:

Execution Time (microseconds): 17.6L + 14.6m + 11.4, where L = # of times CURRMAX changes; M = # of times CURRMAX does not change, L+M = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	DP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, F0, F1.

Algorithm: A loop is set up to compare each element of the array to the current max. Initially, the first element is CURRMAX. Each subsequent element of greater value replaces the former CURRMAX. The counter is decremented after each comparison. The value of CURRMAX when the counter is zero is the max of the array and is passed back to the calling program.

DMIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DMIN Size of Code Area: 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DMIN

Function: Finds minimum value of a double precision scalar array.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:

MIN(<DP scalar array>)

[ ] Other Library Modules:

Execution Time (microseconds): 17.6L + 14.6m + 11.4, where L = # of times CURRMIN changes; M = # of times CURRMIN does not change; L+M = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	DP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, F0, F1.

Algorithm: Similar to MAX functions, except register contains the current minimum and is changed when an element in the array has a smaller value than CURRMIN.

DPROD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DPROD Size of Code Area: 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DPROD

Function: Calculates the product of the elements of a double precision scalar array.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:

PROD(<DP scalar array>)

[ ] Other Library Modules:

Execution Time (microseconds): 20.6n + 6.2 if product is not zero, where n = # of elements in the array. 20.6m + 2.6 if produce is zero, where m = index into the linear representation of the array of the first zero element.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	DP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, F0, F1.

Algorithm: Similar to the algorithm for the SUM functions. An accumulator is initialized to one. The value in the accumulator is multiplied by each element of the array; the result of each multiplication is saved in the accumulator. After each multiplication, the result is checked for a zero product. If the product is ever zero, the routine exits and returns zero.

DSUM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSUM Size of Code Area: 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSUM

Function: Calculates the sum of the elements of a double precision scalar array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
SUM(<DP scalar array>)

☐ Other Library Modules:

Execution Time (microseconds): 7.2 + 11.6n, where n = # of elements in the array.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	DP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, F0, F1.

Algorithm: An accumulator (F0, F1) is initialized to zero. Each element of the array is added to the accumulator in a loop based upon the array size.

EMAX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EMAX Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EMAX

Function: Finds maximum value in a single precision scalar array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
MAX(<SP scalar array>)

☐ Other Library Modules:

Execution Time (microseconds): 9.8 + 10.8m + 12.2L, where m = # of times CURRMAX does not change; L = # of times CURRMAX changes; and M+L = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	SP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, F0, F1.

Algorithm: Same as DMAX except that operations are all single precision.

EMIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EMIN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EMIN

Function: Finds minimum value in an array of single precision scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
MIN(<SP scalar array>)

☐ Other Library Modules:

Execution Time (microseconds): 9.8 + 10.8m + 12.2L, where m = # of times CURRMIN does not change; L = # of times CURRMIN changes; and M+L = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	SP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, F0, F1.

Algorithm: Same as DMIN, except operations are in single precision floating point.

EPROD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EPROD Size of Code Area: 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EPROD

Function: Calculates product of elements of a single precision scalar array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
PROD(<SP scalar array>)

☐ Other Library Modules:

Execution Time (microseconds): 13.2n + 4.6 if produce is not zero, where n = # of elements in the array. 13.2m + 1.4 if product is zero, where m = index into the linear representation of the array of the first zero element.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	SP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, F0, F1.

Algorithm: Same as DPROD.

ESUM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ESUM Size of Code Area: 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ESUM

Function: Calculates sum of elements in a single precision scalar array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
SUM(<SP scalar array>)

☐ Other Library Modules:

Execution Time (microseconds): 5.2 + 6.6n, where n = # of elements in the array.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	SP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, F0, F1.

Algorithm: Same as DSUM.

HMAX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: HMAX Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: HMAX

Function: Finds maximum value in a single precision integer array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
MAX(<SP integer array>)

☐ Other Library Modules:

Execution Time (microseconds): 11.0 + 7.8m + 9.2k, where m = # of times CURRMAX does not change; k = # of times CURRMAX changes; and m+k = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	SP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, R6.

Algorithm: Same as DMAX, except that all operations deal with halfword integers.

HMIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: HMIN. Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: HMIN

Function: Finds minimum value in a single precision integer array.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
MIN(<SP integer array>)

[ ] Other Library Modules:

Execution Time (microseconds): 11.0 + 7.8m + 9.2k, where m = # of times CURRMIN does not change; k = # of times CURRMIN changes; and m+k = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	SP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, R6.

Algorithm: Same as DMIN, except that operations are for halfword integers.

HPROD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: HPROD Size of Code Area: 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: HPROD

Function: Calculates product of elements in a single precision integer array.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
PROD(<SP integer array>)

[ ] Other Library Modules:

Execution Time (microseconds): 12.4n + 5.8 if product is not zero, where n = # of elements in array. 12.4m + 2.2 if product is zero, where m = index into the linear representation of the array of the first zero element.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	SP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, R6.

Algorithm: Same as DPROD.

HSUM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: HSUM Size of Code Area: 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: HSUM

Function: Calculates sum of elements in a single precision integer array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

SUM(<SP integer array>)

☐ Other Library Modules:

Execution Time (microseconds): 4.4 + 5.4n, where n = # of elements in the array.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	SP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: Register Unsafe Across Call: R2, R4, R5, R6.

Algorithm: Same as DSUM.

IMAX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IMAX Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IMAX

Function: Finds maximum value in a double precision integer array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

MAX(<DP integer array>)

☐ Other Library Modules:

Execution Time (microseconds): 11.1 + 7.8m + 4.3k, where m = # of times CURRMAX does not change, k = # of times CURRMAX changes, and m+k = # of elements in array-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	DP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, R6.

Algorithm: Same as DMAX, except that all operations are on fullword integers.

IMIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IMIN Size of Code Area: 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IMIN

Function: Finds minimum value in a double precision integer array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
MIN(<DP integer array>)

☐ Other Library Modules:

Execution Time (microseconds): 11.1 + 7.8m + 9.3k, where m = # of times CURRMIN does not change, k = # of times CURRMIN changes, and m+k = # of elements in array -1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	DP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, R6.

Algorithm: Same as DMIN, except that all operations are done for fullword integers.

IPROD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IPROD Size of Code Area: 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IPROD

Function: Calculates product of elements in a double precision integer array.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
PROD(<DP integer array>)

☐ Other Library Modules:

Execution Time (microseconds): 17.0L + 21.6m + 5.8 if product is not zero, where L = # of positive intermediate products; m = # of negative intermediate products; L+m = # of elements in array. 17.0L + 21.6m + 19.6 if product is not zero, where L and m are as above, L+m = (index into linear representation of the array of the first zero element)-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	DP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, R6, R7.

Algorithm: Same basic algorithm as DPROD, however, special detection of an overflow condition is performed: For fullword integer multiplication, the overflow indicator is only set when -1 is multiplied by -1. The result after each multiplication is checked for an overflow by testing the first 32 bits of the 64 bit result for all zeros or ones. If the result does overflow 32 bits, then a fixed point overflow is forced by adding a very large number to the first register of the pair (the register with the overflowing bits).

ISUM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ISUM Size of Code Area: 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ISUM

Function: Calculates sum of elements in a double precision integer array.

Invoked by: .

[x] Compiler emitted code for HAL/S construct of the form:
SUM(<DP integer array>)

[ ] Other Library Modules:

Execution Time (microseconds): 4.4 + 5.4n, where n = # of elements in array.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	DP	R2 → 0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R2, R4, R5, R6.

Algorithm: Same as DSUM.

5.3.6 Miscellaneous Routine Descriptions

This subsection presents those routines which do not fall easily into the previous five sections. These encompass conversion routines as well as "service" routines used by other library members.

BTOC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: BTOC Size of Code Area: 28 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: BTOC

Function: Conversion from bit data to character data.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
$CHARACTER_{@BIN}$(<bit string>).

[ ] Other Library Modules:

Execution Time (microseconds): 161.0 (for 16-bit string)

Input Arguments:

Type	Precision	How Passed	Units
Bit	-	R5	-
Integer(length)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Character	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7.

Algorithm: First, unwanted bits are shifted out of the string, using the length argument. Then, bits are shifted one by one out of the top of the R5 into the bottom of R4, where they are shifted to bit positions 15 and 31 and converted to character format. The output string is stored halfword by halfword, with the length taken directly from the input length.

CSHAPQ

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CSHAPQ Size of Code Area: 40 Hw

Stack Requirement: 18 Hw Data CSECT Size: 4 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: CTOH,CTOI,CTOE,CTOD

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CSHAPQ

Function: Shapes arrayed character data to arrayed numeric data of an explicit type and precision.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
$INTEGER_N$(CA), $INTEGER_{@DOUBLE,N}$(CA), $SCALAR_N$(CA), $SCALAR_{@DOUBLE,N}$(CA), where CA is a character array of length n of CHARACTER(m).

[ ] Other Library Modules:

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
character array(n)	-	R4 → 1st element	-
integer(n)	SP	R5	-
integer(type *)	SP	R6	-
integer(m)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
array(n)	type	R2	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5

type*: 0 → H conversion
1 → I conversion
2 → E conversion
3 → D conversion

Algorithm:

The address of one of 4 internal loops is loaded from a table using the type code as an index and control is passed to that loop. The four internal loops are similar in action: a call is made to the appropriate character conversion routine (CTOH, CTOI, CTOE, CTOD) followed by the appropriate store (STH, ST, STE, STED) into the result array, followed by instructions to bump both the character and result array pointers, looping on n.

CSHAPQ

Execution Time (microseconds):

For halfword INTEGER conversion:

$$25.2 + \sum_{k=1}^{n} (19.6 + CTOH_K)$$

where $CTOH_K$ = time in CTOH for the Kth conversion.

For fullword INTEGER conversion:

$$24.8 + \sum_{k=1}^{n} (20.2 + CTOI_K)$$

where $CTOI_K$ = time in CTOI for the Kth conversion.

For fullword SCALAR conversion:

$$25.2 + \sum_{k=1}^{n} (19.6 + CTOE_K)$$

where $CTOE_K$ = time in CTOE for the Kth conversion.

For double-word SCALAR conversion:

$$22.8 + \sum_{k=1}^{n} (22.8 + CTOD_K)$$

where $CTOD_K$ = time in CTOD for the Kth conversion.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CSLD Size of Code Area: 246 Hw

Stack Requirement: 22 Hw Data CSECT Size: 4 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CSLD

Function: Loads bit pattern of a character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
B=SUBBIT(C) where B is a bit string and C is a character string.

☐ Other Library Modules:

Execution Time (microseconds): if length(C) = 0: 28.8
If length(C) > 0: 56.3 + 0.8 (if length(C) > 4)

Input Arguments:

Type	Precision	How Passed	Units
char string	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
bit string	length 32	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments: If input string is null, the 0 bit string is returned; no error.

Registers Unsafe Across Call: R5,F0,F1.

Algorithm: The first character is set to 1 by clearing R5. The character width is set to the current length of the string. For the rest, see the description under entry CSLDP, after the character partition checking, at the point marked [A].

Secondary Entry Name: CSLDP

Function: Loads bit pattern of a partitioned character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
BIT_STRING = SUBBIT(CHAR_STRING$_{\text{WIDTH AT FIRST}}$)

☐ Other library modules:

Execution Time (microseconds): 69.7 + 0.8 (if WIDTH > 4)
+ 4.0 (if FIRST is even)

Input Arguments:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-
integer(first character)	SP	R5	-
integer(last character)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
bit string	length(32)	R5	-

Errors Detected:

Error #	Cause	Fixup
17	Character subscript out of legal range.	1) If <1, set to 1 2) IF > length of string, set to length (string) 3) If last char < first char, return 0 string.

Comments: 0 bit string returned if last character specified < first character specified (with ERROR 17). Registers Unsafe Across Call: R5,F0,F1.

Algorithm: The character partition is checked for validty before anything else is done. All error 17's are sent during this phase. [A] The partition width is checked, and if it is ≤ 0, the zero string is returned in R5. If greater than 4, it is set to 4. The address of the halfword containing the first character of the partition is found by adding ½(1+first character) to the address of the first halfword of the string. This halfword and the next two halfwords are loaded into the low half of R4, and the high and low halves of R5, respectively. Unwanted bits are masked off the left and shifted off the right (shift count = 48-8*width), and the desired bit string is left in R5.

Secondary Entry Name: CPSLD

Function: Loads specified bits of a character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
BIT_STRING = SUBBIT$_{\text{FIRST TO LAST}}$(CHAR_STRING)

☐ Other library modules:

Execution Time (microseconds): 71.8

Input Arguments:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-
integer(first bit)	SP	R5	-
integer(last bit)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
bit string	length(32)	R5	-

Errors Detected:

Error #	Cause	Fixup
30	Subbit partition out of range.	1) If 1st bit <1, set to 1 (keep constant partition width by adjusting last bit) 2) If 1st or last bit > last bit of character string, set equal to last bit of char string.

Comments: If input string is null, ERROR 30 is sent and the 0 bit string is returned.
Registers Unsafe Across Call: R5,F0,F1.

Algorithm: The subbit partition is tested for validity before anything else is done. All ERROR 30's are sent during these tests. 4 halfwords containing the required partition are loaded into a register pair. Unwanted bits are shifted off the top (left shift count = first bit-1), and the bottom (right shift count = 64-width), leaving the required string in R5.

Secondary Entry Name: CPSLDP

Function: Loads selected bits of a partitioned character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
BIT_STRING = SUBBIT$_{A\ TO\ B}$(CHAR_STRING$_{C\ TO\ D}$)

☐ Other library modules:

Execution Time (microseconds): 98.6 + 9.2 (if C is even)

Input Arguments:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-
integer(C)	SP	R5	-

(Continued on bottom of this page)

Output Results:

Type	Precision	How Passed	Units
bit string	length (32)	R5	-

Errors Detected:

Error #	Cause	Fixup
17	character subscript out of legal range.	(See CSLDP)
30	subbit partition out of legal range.	(See CPSLD)

Comments:
Registers Unsafe Across Call: R5,F0,F1.

Algorithm: The character partition is checked for validity, and the 0 bit string is returned if last character < first character. Then the subbit partition is checked, resetting the bit pointer to point 8 bits farther on if the character partition begins in the second character of a halfword. 4 halfwords containing the required partition are loaded into register pair R4-R5. Unwanted bits are shifted off the top (shift count = relative 1st bit-1) and the bottom (shift count = 64-bit width), leaving the desired string in R5.

Input Arguments (Con't):

Type	Precision	How Passed	Units
integer(D)	SP	R6	-
integer(A||B)	(SP||SP)	R7	-

Secondary Entry Name: CSST

Function:
Stores a bit string into a character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
SUBBIT(C) = B where B is a BIT string and C is a character string.

☐ Other library modules:

Execution Time (microseconds): if length(C) =0: 26.6
if length(C) > 0: 135.8 + 1.0 (if length(C) > 4)

Input Arguments:

Type	Precision	How Passed	Units
bit string	-	R4	-

Output Results:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
No errors detected in CSST.		

Comments: If the length of the input character string is 0, no error is given, and nothing is changed. CSST cannot change the length of the input string. Registers Unsafe Across Call: R5,F0,F1.

Algorithm:
The first character is set to 1 by clearing R5, the character width is set to the current length of the string. Processing continues as at [A] in the description of the algorithm at entry CSSTP.

Secondary Entry Name: CSSTP

Function: To store a bit string into a partitioned character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
SUBBIT(CHAR_STRING$_{A\ TO\ B}$) = BIT_STRING

☐ Other library modules:

Execution Time (microseconds): 148 + KA + KB, where KA = 1.0 if B-A>4
0 otherwise
KB = 9.2 if A is even
0 otherwise

Input Arguments:

Type	Precision	How Passed	Units
bit string	-	R4	-
integer(first character)	SP	R5	-
integer(last character)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	character subscript out of legal range.	(See CSLDP), except if last char < first char, then leave input string unchanged.

Comments: Registers Unsafe Across Call: R5,F0,F1.
CSSTP cannot change the current length of the character string: gives error is subscript out of current legal range.

Algorithm:

The character partition is checked for validity, and ERROR 17 is sent if anything is bad.

[A] The character partition width is checked. If it is ≤ 0, then the input character string is returned unchanged. If > 4, then it is set to 4. The first bit and last bit are determined as:

First bit = 1 + 8*(first character-1)
Last bit = First bit + 8*character width-1

The first bit, last bit, and character width of the string are then sent to [B] under entry CPSST.

Secondary Entry Name: CPSST

Function: To store a bit string into specified bits of a character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
SUBBIT$_{A\ TO\ B}$(CHAR_STRING) = BIT_STRING

☐ Other library modules:

Execution Time (microseconds): 114.4

Input Arguments:

Type	Precision	How Passed	Units
bit string	-	R4	-
integer(first bit)	SP	R5	-
integer(last bit)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
30	subbit partition out of legal range.	(See CPSLD)

Comments: CPSST cannot change the current length of the input character string. In particular, a null character string input will result in a null string output. Registers Unsafe Across Call: R5,F0,F1.

Algorithm: Set character partition width to the current length of the character string. If it is 0, exit immediately after sending ERROR 30.

[B] Test subbit partition for validity, and send ERROR 30 if anything is bad. Find the first halfword containing the specified partition. The first bit relative to that halfword, and the bit partition width thus:

bit width = last bit - first bit + 1
first halfword = 1 + (first bit - 1)/16
relative first bit = first bit - 16(first halfword - 1)

Load 4 halfwords, beginning with the first halfword of the partition, into register pair R4-R5.

Prepare a mask with 0's in the specified bit positions and 1's else where as the 1's complement of:

$$(2^{\text{bit width}}-1)\,(2^{64-\text{relative last bit}})$$

where relative last bit = relative first bit + bit width - 1.

Use this mask to mask out the old bits in R4-R5. Shift the input bit string left by (64-relative last bit) positions to align it with the specified bit positions. Then OR it into the contents of R4-R5. Store this back into the character string, and exit.

Secondary Entry Name: CPSSTP

Function: To store a bit string into selected bits of a partitioned character string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$SUBBIT_{A\ TO\ B}(CHAR\_STRING_{C\ TO\ D}) = BIT\_STRING$

☐ Other library modules:

Execution Time (microseconds): 145.0 + 9.2(if C is even)

Input Arguments:

Type	Precision	How Passed	Units
integer (C)	SP	R5	-
integer (D)	SP	R6	-

(Continued on bottom of this page)

Output Results:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	character subscript out of legal range	(See CSSTP)
30	subbit partition out of legal range	(See CPSST)

Comments: CPSSTP cannot change the current length of the input character string. Registers Unsafe Across Call: R5,F0,F1.

Algorithm: Check character partition for validity, give any error 17's necessary, and exit if last char < first char on currlen = 0. Reset character pointer to 1 halfword before the first halfword of the specified partition, bumping first and last bits by 8 if the first character is even (so lies in low-order 8 bits of the halfword) after checking validity of first bit, and sending error 30 if it is < 1. Then continue as (B) of CPSST.

Input Arguments (Continued)

Type	Precision	How Passed	Units
integer(A||B)	(SP||SP)	R7	-
bit string	-	R4	

CSTRUC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CSTRUC **Size of Code Area:** 12 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CSTRUC

Function: Compares two structures and returns result (= or ≠) in condition code.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
If S1 = S2 THEN... , where S1 and S2 are structures.

☐ Other Library Modules:

Execution Time (microseconds): 5.4 + 10.4n, n = # halfwords compared.

Input Arguments:

Type	Precision	How Passed	Units
Structure	-	R2 → 1st HW	-
Structure	-	R3 → 1st HW	-
Integer(count)	SP	R5	HW

Output Results:

Type	Precision	How Passed	Units
equal/not equal	-	condition code	-

Errors Detected:

Error #	Cause	Fixup

Comments: In order that the correct code is in the C.C. on exit, it is reset immediately before branching back to the calling program (BCRE does not set the C.C.). An exclusive OR of a register with itself sets to C.C. to '00'(=). An OR of a non-zero register (R4 is used because, as the return address register, it is always assumed to be non-zero) resets to C.C. to '11'(≠).
Registers Unsafe Across Call: R2,R3,R4,R5,R6.

Algorithm: The two structures are compared halfword by haflword until a pair does not match, or all of the halfwords are compared and found to be equal. The condition code is set by the compare instruction.

CTOB

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTOB Size of Code Area: 32 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☑ Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CTOB

Function: Conversion from character string data to bit string.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$BIT_{@BIN}(C)$, where C is a character string.

☐ Other Library Modules:

Execution Time (microseconds): $25.8 + \sum_{k=1}^{NCHAR} (27.8 + KA_k + KB + KC)$

where: (Continued on bottom of this page)

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Bit	length=31 implicitly	R5	-

Errors Detected:

Error #	Cause	Fixup
29	Input string not in standard format.	Return zero bit string.

Comments: If the input string includes more than 32 digits, then high-order bits will be lost. Null string input causes an error.
Registers Unsafe Across Call: R5,F0,F1.

Algorithm: Characters are examined one by one. Blanks are ignored. When a '1' is encountered, a '1' bit is shifted into the low-order bit of the result register. When a '0' is encountered, a '0' bit is shifted into the low-order bit of the result register.

Execution Time (Continued):

NCHAR = length(C)

KA_X = 1.2 if X is odd
0 if X is even

KB = 6.0 if C$(K)='1'
0 otherwise

KC = 4.4 if C$(K)=blank
0 otherwise.

CTOE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTOE Size of Code Area: 287 Hw

Stack Requirement: 30 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CTOE

Function: Performs internal character to single precision scalar conversion.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
SCALAR(<character string>).

☒ Other Library Modules:
CSHAPQ

Execution Time (microseconds): See end of algorithm description.

Input Arguments:

Type	Precision	How Passed	Units
character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
20	Input string not in standard format.	Return 0.0

Comments:
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: First, leading and trailing blanks are stripped from the input string, and an error is signalled if the string has length = 0 or consists entirely of blanks. Next, a scalar is constructed from the digits of the input string to the left of the exponent. The construction proceeds as follows:

First, we set $S_0 = 0$. Now, at the k-th step, $k \geq 1$, we let $S_k = 10*S_{k-1} + d_k$, where d_k is the k-th digit in the string. All calculations are double scalar. When S_k becomes $\geq$ X'4E19999A', all further digits are insignificant and are scanned for validity but otherwise ignored.

(Continued on next page)

CTOE

Algorithm (Con't)

This yields a scalar which may be incorrect by a power-of-10 multiple, but otherwise represents the decimal number of the left of the exponent. As for the power of 10, if a decimal point is encountered while scanning the input string, a count is kept of how many digits there are to the right of the decimal point in the input string. The negative of this count is stored in temporary location COUNTE for later use.

Next, the type of exponent (if any) is determined, and the value is calculated with a simple fixed-point calculation (ab_{10} = 10a + b) and added to COUNTH, COUNTB, or COUNTE accordingly as the type of exponent is hexadecimal, binary, or decimal. Continue this process as long as there are remaining exponents.

If the end of the string is reached with no invalid characters found, then the scalar is modified according to the exponents already computed. First, the power-of-2 exponents are combined as 4H + B, since

$$16^H \cdot 2^B = 2^{4H + B}.$$

The high part of this (power of 16) is added to the exponent of the scalar, while the low 2 bits control a loop in which the scalar is doubled 0-3 times.

Next, the decimal exponent, which has been combined with the correction for the decimal point in the input, is used as a power of 10 in the standard way of taking integral powers. The scalar intermediate is multiplied or divided by this result according to the sign of the exponent, completing the conversion.

CTOE

Timing:

88.4 + 11.0 · (floor (# leading blanks/2))
+ 12.0 (if # leading blanks odd)
+ 10.2 (# trailing blanks)
+ 2.0 (if + sign)
+ 7.0 (if - sign)
+ 59.6 (# significant digits where S_k < X'4E19999A')
+ 17.6 (# significant digits)
+ 47.2 (if at least 1 significant digit)
+ 62.4 (# significant digits where S_k > X'4E19999A')
+ 20.6 (if decimal point)
+ 9.6 (if no exponents of any kind)
+ 40.2 (if any exponents)
+ 9.6·(# E type exponents)
+ 15.2·(# H type exponents)
+ 18.2·(# B type exponents)
+ 9.8·(# exponents)
+ 37.8·(# additional exponents)
+ 0.2·(# exponents with '+' sign)
+ 7.8·(# exponents with '-' sign)
+ 24.6·(total number of exponent digits)
+ 22.8 (if any B or H exponent)
+ 7.6 (total B exponent mod 4)
+ 14.0 (if p=0)

$$\left\{\begin{array}{l} +(17.8 + 27.8\ \mathrm{div}(|p|, 23))\ \text{(if p positive)} \\ +(18.8 + 28.8\ \mathrm{div}(\ p\ , 23))\ \text{(if p negative)} \\ +\ 23.2\cdot(\text{(\# significant zeroes in the binary representation of } |p| \bmod 23) - 1\ \text{(if } |p| \bmod 23 \text{ is even)} \\ +\ 36.2\cdot(\text{(\# significant ones in the binary representation of } |p| \bmod 23) - 1\ \text{(if } |p| \bmod 23 \text{ is odd)} \\ +\ 14.2\ \text{(if } |p| \bmod 23 \neq 0) \\ +\ 28.0 \\ +\ 1.6\ \text{(if p < 0)} \end{array}\right. \qquad \text{(if } p \neq 0)$$

+ 14.4·((# of even indexed characters after leading blanks(+1)
+ 15.6·((# odd indexed characters after leading blanks)+1)

where p = total of E type exponents - (# significant digits after decimal point).

Secondary Entry Name: CTOD

Function: Performs internal character to double precision scalar conversion.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form: $SCALAR_{@DOUBLE}$(<character string>).

[X] Other library modules: CSHAPQ

Execution Time (microseconds): Time is computed by CTOE formula - 1.8.

Input Arguments:

Type	Precision	How Passed	Units
character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Errors Detected:

Error #	Cause	Fixup
20	Input string not in standard format.	Return 0.0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Same routine as CTOE; all conversions result in a double precision value of which the portion in F1 is discarded when single precision is desired by the caller of this routine.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTOI **Size of Code Area:** 104 Hw

Stack Requirement: 20 Hw **Data CSECT Size:** 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CTOI

Function: Converts a character string to a double precision integer.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form: $INTEGER_{@DOUBLE}$(<character string>) or $BIT_{@DEC}$(<character string>)

[X] Other Library Modules: CSHAPQ

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
22	Input string not in standard format.	Return 0

Comments:

Registers Unsafe Across Call: R5,F0,F1.

Algorithm: First, leading blanks are stripped from the input string. If a minus sign is encountered, a flag is set. The basic conversion proceeds as follows: Initialize i_0 to 0. At step k, $k \geq 1$, let $i_k = 10.1_{k-1} + d_k$, where d_k is the k-th digit in the input string. At the end, this fixed-point calculation gives a fullword integer. This sign is tacked on, and the result is shifted left 16 bits if a halfword answer is required.

CTOI

Exeuction Time (microseconds):

k + 11.0·(floor (# leading blanks/2))
+ 18.6 (if # leading blanks odd)
+ 9.4 (if '-' sign)
+ 10.6 (if first character is a number)
+ 15.6·(# even index characters after leading blanks)
+ 14.4·(# odd index characters after leading blanks)
+ 13.0 (if # trailing blanks > 0)
+ 8.4·(# trailing blanks)
+ 28.2·((# non blank characters) -1)

where k = 72.6

Secondary Entry Name: CTOH

Function:
Converts a character string to a single precision integer.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: INTEGER @SINGLE (<character string>).

☑ Other library modules: CSHAPQ

Execution Time (microseconds): Same as for CTOI, except k = 74.4.

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
22	Input string not in standard format.	Return 0

Comments:

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

See CTOI.

Secondary Entry Name: CTOK

Function: Converts a character string to a 32-bit string for use with the @DEC of the BIT conversion function.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: $BIT_{@DEC}$(<character string>).

☐ Other library modules:

Execution Time (microseconds): (See below).

Input Arguments:

Type	Precision	How Passed	Units
character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
bit string	32-bits	R5	-

Errors Detected:

Error #	Cause	Fixup
22	Input string notin standard format.	Return 0

Comments: Registers Unsafe Across Call: R5,F0,F1.

Algorithm: See CTOI.

Execution Time: 85.8 + 11.0·(floor (# leading blanks/2))
+ 18.6 (if # leading blanks odd)
+ 15.6·(# even index characters after leading blanks)
+ 14.4·(# odd index characters after leading blanks)
+ 13.0 (if # trailing blanks > 0)
+ 8.4·(# trailing blanks)
+ 28.2·(# non blank characters - 1)

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTOX **Size of Code Area:** 58 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 4 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CTOX

Function: Conversion from character string to bit string, hexadecimal radix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form: $BIT_{@HEX}$(C), where C is a character string.

☐ Other Library Modules:

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Bit	32 bits	R5	-

Errors Detected:

Error #	Cause	Fixup
32	String not in standard hexadecimal conversion format.	Return 0

Comments: Imbedded blanks,or leading or trailing blanks, are all considered invalid characters. An input string too long to be accommodated in 32 bits will cause high order bits to be lost. Registers Unsafe Across Call: R5,F0,F1.

Algorithm: Characters are fetched one by one. In the CTOX entry, characters 'A' - 'F' are converted to their bit equivalents, characters '0' - '9' are passed on to the common section, and an error is signalled if the input character lies between X'39' and X'41' (DEU characters '9' and 'A' respectively). In the CTOO entry, an error is sent if the character is greater than X'37' (DEU character '7'). Other characters are passed to the common sechon for translation.

In the common section, decimal digits 0-9 (0-7 for octal) are translated to their bit equivalents, and an error is sent if the character precedes '0' in the collating sequence. These bit equivalents, and the ones passed from the CTOX section, are shifted into the low-order 4 bits (3 for octal) of the result register.

CTOX

Execution Time (microseconds):

$$32.0 + \sum_{k=1}^{NCHAR} (33.6 + KA_k + KB_k)$$

where:

NCHAR = length(C)

KA_X = 0 if C$(X) is alphabetic
6.8 if C$(X) is numeric

KB_X = 1.2 if X is odd
0 if X is even

Secondary Entry Name: CTOO

Function: Conversion from character string to bit string, octal radix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$BIT_{@OCT}(C)$, where C is a character string.

☐ Other library modules:

Execution Time (microseconds): $33.4 + \sum_{k=1}^{NCHAR} (34.2 + KA_k)$, where

NCHAR = length(C),

KA_X = 1.2 if X is odd
0 otherwise

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2 → descriptor	-

Output Results:

Type	Precision	How Passed	Units
Bit	32-bits	R5	-

Errors Detected:

Error #	Cause	Fixup
31	String not in standard octal conversion format.	Return 0

Comments:

Registers Unsafe Across Call: R5,F0,F1.

Algorithm: See CTOX.

DSLD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSLD Size of Code Area: 22 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSLD

Function: Loads specified bits of a double precision scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

SUBBIT(SCALAR_VAR@DOUBLE); SUBBIT$_{\text{FIRST TO LAST}}$(SCALAR_VAR@ DOUBLE);

☐ Other Library Modules:

Execution Time (microseconds): 36.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	R2 → Scalar	-
Integer(first bit)	SP	R5	-
Integer(last bit)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Bit string	length 32	R5	-

Errors Detected:

Error #	Cause	Fixup
30	Subbit partition out of range.	1) If first bit < 1, set to 1 2) If last bit > 64, set to 64

Comments:

Registers Unsafe Across Call: R5.

Algorithm: Get the double word operand in register pair R2-R3. If first bit -1 < 0, then given ERROR 30 and set to 0. Use first bit 1 as left shift count to eliminate unwanted high order bits. Compute 64 - last bit + first bit -1, and give ERROR 30 and set to 0 if it is < 0. Use this as right shift count to justify bit string in R3. Return contents of R3.

DSST

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSST Size of Code Area: 54 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSST

Function: Stores a bit string into selected bits of a double word scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

SUBBIT$_{\text{A TO B}}$(DOUBLE_SCALAR) = BIT_STRING

☐ Other Library Modules:

Execution Time (microseconds): 64.6

Input Arguments:

Type	Precision	How Passed	Units
Integer(A)	SP	R5	-
Integer(B)	SP	R6	-
Bit string	-	R7	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	R2 → scalar	-

Errors Detected:

Error #	Cause	Fixup
30	Subbit partition illegal.	1) If first bit < 1, set to 1 2) If last bit > 64, set to 64

Comments:

Registers Unsafe Across Call: None.

Algorithm: Check first bit. If < 1, send error 30 and set to 1. Save the last bit, and get the partition width as last bit - first bit + 1. Create a mask of width = partition width as $2^{\text{width}} - 1$. If last bit ≤ 64, shift left by 64 - last bit. If last bit > 64, send error 30 and set last bit to 64 by shifting right by last bit - 64. Then invert the (doubleword) mask. Mask out the selected bits of the operand in storage. Then, shift the input bit string to the right position (left 64 - last bit, or right last bit - 64), and OR to the operand in storage, completing the operation.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ETOC Size of Code Area: 278 Hw

Stack Requirement: 20 Hw Data CSECT Size: 64 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ETOC

Function: Converts a single precision scalar to standard internal character-string format for a scalar.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

CHARACTER(SCALAR_VAR)

☐ Other Library Modules:

Execution Time (microseconds): 336.9

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Character string	length=14	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments: maxlength of output string is ignored.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Clear F1 to convert to double precision. Determine the sign and get the absolute value of the input. If input = 0, output string is '0.0' padded with blanks to length 14 (length 23 for DTOC).

The next operation reduces the exponent of the scalar to X'41' keeping track of the change in exponent that this requires. Since $\log_{10}x = (\log_{10}16)(\log_{16}x)$, this is done by getting (exponent - 65)*$\log_{10}16$ and using this as an exponent of 10, dividing the scalar by the result. It is possible for this to be off by 1, so another pass is made before continuing. At this point, the number is between 1 and 16. If it is greater than or equal to 10, multiply by 1/10 and record the exponent as one greater.

(Continued on next page)

ETOC

Algorithm (Con't)

This causes the first decimal digit of the number to be the first hexadecimal digit of the scalar, in bits 8-11 of F0. This is stored, together with a blank if the value is ≥ 0, or a '-' if the value < 0. The remaining mantissa is in fractional form in F0-F1. This hexadecimal fraction is converted to decimal digit-by-digit by successive multiplication by 10. One digit is generated and stored with the decimal point, then 6 digits are stored in the next three halfwords.

The sign of the exponent (as calculated above) is tested, and either 'E+' or 'E-' is stored in the next halfword. Two decimal digits of exponent are stored in the last halfword.

ETOC

Secondary Entry Name: DTOC

Function: Converts a double precision scalar to standard internal character-string format for scalar.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:

CHARACTER(DOUBLE_SCALAR)

[ ] Other library modules:

Execution Time (microseconds): 602.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0-F1	-

Output Results:

Type	Precision	How Passed	Units
Character string	length=23	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm: Similar to ETOC, except of course that F1 is not zeroed. Also, rather than 6 digits being stored in the loop, 14 are computed and stored. The exponent section also looks different, as one more digit is stored with the exponent, changing its alignment, thus storing successively '<digit>E', '±<digit>', '<digit><garbage>' in the last 3 halfwords.

ETOH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ETOH **Size of Code Area:** 14 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

[X] Intrinsic Sector 0 [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ETOH

Function: Converts single precision scalar value to single precision integer.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
I = S; where I is a single precision integer, and
S is a single precision scalar.

[X] Other Library Modules: QSHAPQ

Execution Time (microseconds): 15.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm: The six most significant hex digits of the scalar argument are converted to a fullword integer value. The 4 most significant hex digits of the integer value are left in the top halfword of the fixed point register after rounding the fraction.

ETOH

Secondary Entry Name: DTOH

Function: Converts a double precision scalar value to a single precision integer.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

I = D; where I is a single precision integer, and
D is a double precision scalar.

☐ Other library modules:

Execution Time (microseconds): 17.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm: See ETOH.

GTBYTE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: GTBYTE **Size of Code Area:** 14 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: GTBYTE

Function: Fetches one character from a character string. Used for character manipulation by other library routines.

Invoked by:

☐ Compiler emitted code for HAL/S construct of the form:

☒ Other Library Modules: CPAS, CTOE, CLJSTV, CINDEX, CRJSTV, CTOB, CTOI, CTOX, CTRIMV.

Execution Time (microseconds): 14.4 to obtain lower byte
15.6 to obtain upper byte

Input Arguments:

Type	Precision	How Passed	Units
pointer	-	R2 →1HW in front of HW to fetch from	-
flag (which byte to fetch)	-	lower half of R2 { 00-upper byte; X'8000'-lower byte }	-

Output Results:

Type	Precision	How Passed	Units
single character	-	R5 - upper halfword	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0,F1.

Algorithm: The halfword off of the pointer is loaded into a register for manipulation. If the flag indicates the upper byte is requested, the register is shifted right 8 bits and the lower half of the register is cleared to leave only the desired byte in the upper halfword of the register. If the flag indicates the lower byte is requested, then the first byte of the register is cleared. The flag is reset to indicate the upper byte if the lower byte was requested and vice versa, and the pointer is updated if the fetched byte was even. This is done now since GTBYTE is usually called a number of times from a loop.

ITOC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ITOC Size of Code Area: 104 Hw

Stack Requirement: 28 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ITOC

Function: Converts a fullword integer into standard internal character string format for integers.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
CHARACTER(FULLWORD_INTEGER)

☐ Other Library Modules:

Execution Time (microseconds): 254.6

Input Arguments:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments: No leading zeros or leading or trailing blanks, Maxlength of output area ignored. Registers Unsafe Across Call: None.

Algorithm: Digits are generated one by one. Thus: Let I = input integer. Then:

$d_k = I_k - 10(I_k/10)$ integer multiply and divide.

$I_{k+1} = (I_k - d_k)/10.$

The process terminates when $I_k = 0$. As pairs of digits are generated, they are stored, right to left, in a temporary output area. The temporary result is then given a sign if necessary and moved to the output area. If an odd number of characters were generated, the move is with 8 bits offset for left alignment.

Secondary Entry Name: HTOC

Function: Converts a halfword integer into standard internal character-string format for integers.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
CHARACTER(HALFWORD_INTEGER)

☐ Other library modules:

Execution Time (microseconds): 189.6

Input Arguments:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments: No leading zeroes or leading or trailing blanks, Maxlength of output area ignored.
Registers Unsafe Across Call: None.

Algorithm: Shift right algebraic 16 to convert single integer to double. Then proceed as in ITOC.

ITOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ITOD **Size of Code Area:** 20 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

[x] Intrinsic Sector 0 [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ITOD

Function: Converts a double precision integer to a double precision scalar.

Invoked by:

[x] Compiler emitted code for HAL/S construct of the form:
D = I; where D is a double precision scalar, and
I is a double precision integer.

[x] Other Library Modules: QSHAPQ

Execution Time (microseconds): 15.6

Input Arguments:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm: The first register of the result register pair is set to plus or minus 0, depending on the sign of the argument. The absolute value of the argument is loaded into the second register of the pair, and the result is normalized to a double precision scalar by adding D'0'.

ITOE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ITOE **Size of Code Area:** 6 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

[X] Intrinsic Sector 0 [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ITOE

Function: Converts a double precision integer to a single precision scalar.

Invoked by:

[X] Compiler emitted code for HAL/S construct of the form:
S = I; where S is a single precision scalar, and
I is a double precision integer.

[X] Other Library Modules: QSHAPQ

Execution Time (microseconds): 12.0

Input Arguments:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm: The integer argument is converted to floating point by the CVFL instruction and the binary point is adjusted by multiplication by a scale factor.

KTOC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: KTOC Size of Code Area: 70 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: KTOC

Function: Performs bit-string to character conversion with decimal radix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$CHARACTER_{@DEC}(BIT\_STRING)$

☐ Other Library Modules:

Execution Time (microseconds): 262.5 (for 16 bits)

Input Arguments:

Type	Precision	How Passed	Units
Bit string	-	R5	-
integer(length of bit string)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments: Maxlength of output area ignored. No leading or trailing blanks. "Sign bit" of input string ignored.

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm: The length of the character string is computed as:

$$1 + (\log_{10}2)(\text{bit length}) \quad \text{truncated to an integer}$$

A halfword count is computed from this as:

$$\text{halfword count} = (1 + \text{character count})/2$$

Decimal digits are generated one at a time, from right to left, thus: Let I_0 = input string as unsigned integer.

$$d_k = I_k - 10(I_k/10) \quad \text{integer multiply and divide}$$

$$I_{k+1} = (I_k - d_k)/10$$

(Continued on next page)

KTOC

Algorithm (Con't)

The process terminates when the halfword count is reached, with two digits stored per halfword. At the end, if an odd number of characters have been stored, the string must be shifted one character to the left for proper alignment.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MSTRUC **Size of Code Area:** 8 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MSTRUC

Function: Moves a structure.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
S1 = S2, S1 and S2 are structures.

☐ Other Library Modules:

Execution Time (microseconds): 4.2 + 9.4n, n = # halfwords moved.

Input Arguments:

Type	Precision	How Passed	Units
Structure (S2)	-	R2 → first HW	-
Integer (#HW)	SP	R5	HW

Output Results:

Type	Precision	How Passed	Units
Structure (S1)	-	R1 → first HW	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,R6.

Algorithm: The structure is moved halfword-by-halfword in a load, store sequence for the number of halfwords specified in R5.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: QSHAPQ **Size of Code Area:** 74 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: ETOH(DTOH), ROUND(DTOI), ITOE, ITOD

ENTRY POINT DESCRIPTIONS

Primary Entry Name: QSHAPQ

Function: Shapes data of a given type and precision to data of an explicit type and precision.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
Used by the INTEGER, SCALAR, MATRIX, and VECTOR shaping functions.

☐ Other Library Modules:

Execution Time (microseconds): 42.6 + 31.8n, n = # times transferred.

Input Arguments:

Type	Precision	How Passed	Units
Integer/scalar	SP/DP	R2 → 1st HW	-
Integer (flag)	DP	R6: upper half for input data, lower half for output.	-
Integer (count)	SP	R5: number of times to transfer	-

Output Results:

Type	Precision	How Passed	Units
Integer/Scalar	SP/DP	R1 → 1st HW	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion.	Set to maximum representable value.

Comments: QSHAPQ is called only if more than one item of the same data type must be shaped. If only one item must undergo conversion, the conversion functions (DTOI, ETOI, ITOD, HTOE, etc.) are used. (Cont'd. on next page)

Algorithm: The flags for the input and output data are examined. The appropriate 'LOAD' routine is executed to load one item to be shaped. The appropriate 'STORE', or in some cases, 'CONVERT AND STORE' routine stores the shaped data item in the area pointed to by the destination pointer. The source pointer is updated after each load; the destination pointed is updated after each store. Data is shaped and transferred item-by-item.

The values of the flags (R6 upper and lower) are:
0 - HW integer
1 - FW integer
2 - FW scalar
3 - DW scalar

QSHAPQ

Comments: (Continued)

Registers Unsafe Across Call: F0,F1.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: RANDOM Size of Code Area: 46 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: RANDOM

Function: Generates random number with uniform distribution in range (0.0, 1.0).

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
...RANDOM...

☐ Other Library Modules:

Execution Time (microseconds): 54.4

Input Arguments:

Type	Precision	How Passed	Units
None	-	-	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0/F1	-

Errors Detected:

Error #	Cause	Fixup

Comments: The original SEED(F'1435') is declared as a data constant. To allow storage into this "constant" for updating SEED, the storage protection is turned off for SEED. Registers Unsafe Across Call: F0,F1,F2,F3.
Algorithm: Multiply F'65539' by SEED. SEED originally = F'1435', but is updated on each pass through RANDOM. Use the least significant 32 bits of this product (SEED x 65539) to form the new SEED. If the result is $\geq$ 0, then RESULT = new SEED. If RESULT < 0, then new SEED = RESULT-NEGMAX, where NEGMAX = X'80000000'. The positive new SEED is saved for future use, and is also converted to a floating point number for present compuation of a random number. Multiply the floating point value by 2^{-31} to produce a random number in the range (0.0, 1.0).

Secondary Entry Name: RANDG

Function: Generates random number from Gaussian distribution, mean zero, variance one.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
...RANDOMG...

☐ Other library modules:

Execution Time (microseconds): 575.8

Input Arguments:

Type	Precision	How Passed	Units
None			

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0/F1	-

Errors Detected:

Error #	Cause	Fixup

Comments: Same as RANDOM. Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm: RANDG uses the formula $Y = \sum_{i=1}^{12} X_i - 6.0$, where X_i is a random number generated by RANDOM.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: STBYTE **Size of Code Area:** 22 Hw

Stack Requirement: 0 Hw **Data CSECT Size:** 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: STBYTE

Function: Stores one character into a character string; Used for character manipulation by other library routines.

Invoked by:

☐ Compiler emitted code for HAL/S construct of the form:

☒ Other Library Modules: CLJSTV, CPAS, CRJSTV, CTRIMV

Execution Time (microseconds): 19.2 to store in upper byte.
17.2 to store in lower byte.

Input Arguments:

Type	Precision	How Passed	Units
Single character	-	R5	-
flag(which byte to store into)		lower HW of R1 { 00-upper byte, X'3000' lower byte	

Output Results:

Type	Precision	How Passed	Units
pointer	-	R1 → HW to store into	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: R1,R4,R5,F0,F1.

Algorithm: The flag is tested for an even or odd byte to store into. If odd (upper), the flag is set to indicate even (lower) for the probable loop that STBYTE is in. Then, the byte is inserted into the upper byte of the appropriate halfword. If the flag indicates an even byte to store into, then the byte is inserted into the lower byte of the appropriate halfword. The flag is set to 0 to indicate that the next time around in the loop, the byte will be odd. The pointer is updated to the next halfword.

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: XTOC Size of Code Area: 68 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

☒ Intrinsic ☐ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: XTOC

Function: Converts bit string to a string of hexadecimal characters.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$CHARACTER_{@HEX}(BIT\_STRING)$

☐ Other Library Modules:

Execution Time (microseconds): 35.9 + 32.2 ·(# of digits 0-9) + 33.9 · (# of letters A-F), where 8·((# of digits)+(# of letters))= # bits.

Input Arguments:

Type	Precision	How Passed	Units
bit string	-	R5	-
Integer(length of bit string)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments: Output string length depends on input string length. The maxlength of the output area is ignored.

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm: A character count is determined as the integer part of (bit length + 3)/4. The bit string is positioned in register pair R4-R5 with the first hexadecimal digit in bits 12-15 of R4 thus:

1) Clear R4; string right-justified in R5 on input.
2) Compute greatest multiple of 4 in 52 - bit length.
3) Use result of 2) as a shift count to shift R4-R5 left double.

Compute a halfword count for use as a loop counter:

halfword count = (1 + character count)/2

The character count is stored in the descriptor halfword as the current length of the output string. Digits are generated by shifting left 4 and stored two at a time in the output string, after converting DEU format by adding X'30' to each digit. Exit when proper number of halfwords have been stored.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

Secondary Entry Name: OTOC

Function: Converts a bit string into a string of octal characters.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$CHARACTER_{@OCT}(BIT\_STRING)$

☐ Other library modules:

Execution Time (microseconds): 46.2 + 32.3 ·(# of digits), where 6 · (# of digits) = # bits + 2.

Input Arguments:

Type	Precision	How Passed	Units
bit string	-	R5	-
Integer(length of bit string)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
character string	-	R2 → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments: Output string length depends on input string length. The maxlength of the output area is ignored.

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm: First, a character count is determined as the integer part of (bit length + 2)/3. The bit string is positioned in register pair R4-R5 with the first octal digit in bits 13-15 of R4 as follows: 1) Begin with R4 clear and the string right-aligned in R5. 2) Compute the shift count as 51-3(character count) and 3) shift R4-R5 left double by this amount.

Complete a halfword count for use as a loop counter as:

halfword count = (1 + character count)/2

The character count is stored in the descriptor halfword of the output string. Then, digits are generated in a loop, two at a time, by shifting R4-R5 left double 3 bits and adding X'30' to give the appropriate DEU character. As pairs of digits are assembled, they are stored into the output string, and exit is taken when the proper number of halfwords have been stored.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

5.3.7 REMOTE Routine Descriptions

This subsection describes those routines which perform operations on REMOTE data. REMOTE data is data which may reside in a sector of AP-101 core which is neither sector 0 nor the current data sector indicated in the Program Status Word at the time the routine is called. In order to insure addressability of such data, these routines are passed, instead of pointers directly to their arguments, pointers to complete address constants, or "ZCONS", containing both the address of the argument and the number of the sector in which it resides. These complete address constants, together with a special AP-101 addressing mode, allow access to any area of AP-101 core without changing bits in the Program Status Word.

REMOTE routines are invoked (rather than the normal versions of the same routines) when at least one of the arguments of the routine has the REMOTE attribute. Since this attribute only applies to aggregate data types (VECTOR, MATRIX, STRUCUTRE and CHARACTER types), only these four types of routines have REMOTE versions.

CASRPV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CASRPV **Size of Code Area:** 86 **Hw**

Stack Requirement: 22 **Hw** **Data CSECT Size:** 2 **Hw**

☐ **Intrinsic** ☒ **Procedure**

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CASRPV

Function: Remote character assignment to temporary from partitioned input.

Invoked by:

☒ **Compiler emitted code for HAL/S construct of the form:**

...C2 $_{I\ TO\ J}$... where C2 is a REMOTE character variable and result is a temporary string.

☒ **Other Library Modules:**

CPASRP

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
integer (I)	SP	R5	-
integer (J)	SP	R6	-
character (C2)	-	R4 → ZCON → descriptor	-

Output Results:

Type	Precision	How Passed	Units
character(temporary)	-	R2 → ZCON → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Specified partition outside current string range.	Set bad partition pointers(s) to the limits of the current string.
	Last character indicator less than first character indicator and output string is not null	Return null string.

Algorithm:

Set maxlength of result to 255. Test position of 1st character of partition. If < 1 then send error and set to 1.

Compare position of last character of partition. If it is > than maxlength, reset to maxlength, and send error.

(Continued on next page)

CASRPV

Execution Time (microseconds):

If P = 0 and length(C2) = 0: 76.8

If P > 0 and I is odd: 89.0 + 15.8 · ceiling(n/2)

If P > 0 and I is even: 94.2 + 21.2 · ceiling(n/2)

where p = J - I + 1

n = minimum(p, 255)

Algorithm (Con't)

Compare first and last positions. If last < first, then if input string is null do not send error. If input string is not null, send error and set result to null string. Make sure partition length does not exceed the maxlength of the destination string. If it does, truncate it. Increment character count before dividing by 2 to round resulting halfword count to next highest halfword. If position of first character of partition is odd, then transfer halfword by halfword. Otherwise, it is necessary to line characters up into right halves of halfwords by shifting.

Comments:

Registers Unsafe Across Call: None.

Secondary Entry Name: CASRP

Function:

REMOTE character assignment to declared data, partitioned input.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
C1 = C2 $_{I\ TO\ J}$; where C1 and/or C2 are REMOTE character data.

☐ Other library modules:

Execution Time (microseconds): (See next below).

Input Arguments:

Type	Precision	How Passed	Units
integer (I)	SP	R5	-
integer (J)	SP	R6	-
character(C2)	-	R4 → ZCON → descriptor	-

Output Results:

Type	Precision	How Passed	Units
character(C1)	-	R2 → ZCON → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Same causes and fix-ups as CASRPV	

Comments:

Registers Unsafe Across Call: None.

Algorithm:

Same as CASRPV except maxlength of resultant string is used as passed and not set to 255.

Execution Time:

if p = 0 and length(C2) = 0: 69.4

if p > 0 and I is odd: setup + 15.8 · (ceiling(n/2))

if p > 0 and I is even: setup + 5.2 + 21.2 · (ceiling(n/2))

where p = J - I + 1

setup = 81.6 if p ≥ max length(C1)
82.4 if p > max length(C1)

n = minimum (p, max length(C1))

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CASRV **Size of Code Area:** 36 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CASRV

Function: Remote character assignment to a temporary receiver.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

C1 = C2 where C1 or C2 is a REMOTE character string, C1 is a temporary.

☐ Other Library Modules:

Execution Time (microseconds): if n = 0: 59.6
if n > 0: 60.8 + 12.6 · (ceiling(n/2))
where n = length(C2).

Input Arguments:

Type	Precision	How Passed	Units
character (C2)	-	R4 → ZCON → descriptor	-

Output Results:

Type	Precision	How Passed	Units
character (temporary)	-	R2 → ZCON → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: None.

Algorithm:

Sets maxlength of result to 255. If the current length of the input string > maxlength of result, set current length of result to maxlength. Otherwise, set current length of result to current length of input. Find # of halfwords to move by shifting right 1 # of characters. Move halfword by halfword. If there is an odd # of characters, last byte moved is garbage.

Secondary Entry Name: CASR

Function: Remote character assignment to a non-temporary receiver.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

C1 = C2, where C1 and/or C2 is a remote character string.

☐ Other library modules:

Execution Time (microseconds): if n = 0: 52.6
if n > 0: 51.8 + 12.6 · (ceiling(n/2)) + .8 (if length(C2)> maxlength(C1)), where n = minimum(length(C2), maxlength(C1)).

Input Arguments:

Type	Precision	How Passed	Units
character string	-	R4 → ZCON → descriptor	-

Output Results:

Type	Precision	How Passed	Units
character string	-	R2 → ZCON → descriptor	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: None.

Algorithm:

Same as CASRV, but do not set maxlength of result to 255.

CPASR

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPASR **Size of Code Area:** 132 Hw

Stack Requirement: 24 Hw **Data CSECT Size:** 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CPASR

Function: Remote character assignment to a partitioned receiver.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$C1_{I\ TO\ J}$ = C2, where C1 or C2 is a remote character string.

☒ Other Library Modules:

CPASRP

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
integer(I)	SP	R5	-
integer(J)	SP	R6	-
character (C2)	-	R4 → ZCON → descriptor	-

Output Results:

Type	Precision	How Passed	Units
character(C1)	-	R2 → ZCON → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	index of first character < 1	Set to 1
	index of last character > max length of receiver.	Set to max length.
	index of last character < index of first character.	return receiver unchanged

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

If R5 < 1 then send error and set to 1
If R6 > max length then send error and set to max length
If R6 > curr len of receiver, then update currlen of receiver
If R6 < R5 then send error and exit immediately. Otherwise, move partition character by character.

CPASR

Execution Time (microseconds):

$$77.9 + KA + KB + \sum_{k=1}^{LHP} (6.0 + KC_{LOUT+K}) + KD + \sum_{k=1}^{NCHAR} (8.4 + KE_k + KC_{I+K-1}) + KF + \sum_{k=1}^{RHP} (6.0 + KC_{I+LIN+K-1}) + KG$$

where:

LOUT = length(C1) before assignment

LIN = length(C2)

KA = 0 if J <= LOUT
13.0 if J > LOUT

LPART = J - I + 1

KB = 15.8 if LPART > 0 and LIN <= LPART
12.0 if LPART > 0 and LIN > LPART
0 if LPART = 0

LHP = I-LOUT-1 if I > LOUT + 1
0 otherwise

KC_X = 19.8 if X is odd
20.2 if X is even

KD = 3.2 if LHP = 0 and I is odd
4.2 if LHP = 0 and I is even
1.0 if LHP > 0 and LOUT is odd
0 if LHP > 0 and LOUT is even

NCHAR = minimum(LPART,LIN)

KE_X = 13.8 if X is odd
14.4 if X is even

KF = -.8 if NCHAR > 0
0 if NCHAR = 0

RHP = LPART-LIN if LPART > LIN
0 otherwise

KG = 0 if RHP > 0
.4 if RHP = 0

Note: If any of LHP, NCHAR, RHP is zero, then that respective summation is also zero.

CPASRP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPASRP Size of Code Area: 16 Hw

Stack Requirement: 146 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: CPASR, CASRPV

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CPASRP

Function: Remote character string assignment of partitioned input to partitioned output.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$C1_{i\ TO\ j} = C2_{k\ to\ 1}$ C1, C2 character strings.

☐ Other Library Modules:

Execution Time (microseconds): (See next page).

Input Arguments:

Type	Precision	How Passed	Units
character(C2)	-	R4 → ZCON → descriptor	-
integer(k)	SP	R5	-
integer(l)	SP	R6	-
integer(i||j)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
character (C1)	-	R2 → ZCON → descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Subscript of character string out of bounds.	Set out-of-bounds value to first or last character of associated string.

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Saves pointer to result in work area, loads address of vac in R1, and branches to CASRPV. Returns, loads result address in R1, loads arg 3 and arg 7 in R5 and R6 respectively and branches to CPASR, and returns.

CPASRP

Execution Time (microseconds):

$$132.3 + KA + KB + KC + \sum_{k=1}^{LHP} (6.0 + KD_{OUTLEN+K}) + KE + \sum_{k=1}^{NCHAR} (8.4 + KF_k + KD_{I+K-1}) + KG + \sum_{k=1}^{RHP} (6.0 + KD_{I+INLEN+k-1}) + KH$$

where:

INPART = L-K+1 if L >= K
0 otherwise

INLEN = minimum(INPART, 255)

KA = 76.8 if INPART = 0 and length(C2) = 0
89.0 + 15.8(ceiling(INLEN(2)) if inpart > 0 and K is odd
94.2 + 21.2(ceiling(INLEN(2)) if inpart > 0 and K is even

OUTLEN = length(C1) before assignment

KB = 0 if J <= OUTLEN
13.0 if J > OUTLEN

OUTPART = J - I + 1 if J >= I
0 otherwise

KC = 15.8 if OUTPART > 0 and INLEN = OUTPART
12.0 if OUTPART > 0 and INLEN OUTPART
0 if OUTPART = 0

LHP = I - OUTLEN - 1 if I > OUTLEN + 1
0 otherwise

KD_X = 9.8 if X is odd
20.2 if X is even

KE = 3.2 if LHP = 0 and I is odd
4.2 if LHP = 0 and I is even
1.0 if LHP > 0 and OUTLEN is odd
0 if LHP > 0 and OUTLEN is even

NCHAR = minimum(OUTPART,INLEN)

KF_X = 13.8 if X is odd
14.4 if X is even

KG = -.8 if NCHAR > 0
0 if NCHAR = 0

(Continued on next page).

CPASRP

Execution Time (Continued):

RHP = OUTPART - INLEN if OUTPART > INLEN
0 otherwise

KH = 0 if RHP > 0
.4 if RHP = 0

Note: If any of LHP, NCHAR, RHP is zero, then the respective summation is also zero.

CSTR

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CSTR Size of Code Area: 18 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CSTR

Function: Comparison of REMOTE structures.

Invoked by:
☒ Compiler emitted code for HAL/S construct of the form:
IF S1 = S2... Where S1, S2, or both is a REMOTE structure occupying n halfwords.
☐ Other Library Modules:

Execution Time (microseconds): 22.8 + 14.8n if structures compare, where n = # of halfwords in structure. 19.6 + 14.8n if structures do not compare, where n = index of first non-matching halfwords in structure

Input Arguments:

Type	Precision	How Passed	Units
structure(left comparand)	S1	R2 → ZCON → first Hw	-
structure(right comparand)	S2	R4 → ZCON → first Hw	-
integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
equal/not equal		Condition Code	

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: None.

Algorithm:
Compares structures, halfword by halfword, until two are found that are different or the end of the structure is reached. If inequality is found then set CC to 1 and return. If equal, then set CC to 0 and return.

MSTR

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MSTR Size of Code Area: 10 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MSTR

Function: Moves a structure to or from a remote location.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
S1 = S2..., where S1, S2, or both is a REMOTE structure occupying n halfwords.

☐ Other Library Modules:

Execution Time (microseconds): 16.8 + 15.0n

Input Arguments:

Type	Precision	How Passed	Units
structure (S2)	-	R4 → ZCON → first Hw	-
integer (n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
structure (S1)	-	R2 → ZCON → first Hw	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: None.

Algorithm: Moves structure halfword by halfword.

MRODNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MRODNP Size of Code Area: 16 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MRODNP

Function: Moves a scalar value to all positions of a partition of a double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
$M_{A \text{ TO } B,C \text{ TO } D} = X$; where X is a scalar, and M is a double precision REMOTE matrix.

☐ Other Library Modules:

Execution Time (microseconds): 22.8 + n(5.6 + 9.8m)

Input Arguments:

Type	Precision	How Passed	Units
scalar	DP	F0	-
integer(n)	SP	R5	-
integer(m)	SP	R6	-
integer(outdel)	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
matrix (n,m)	DP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:
Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as MROSNP except use 4 · (# columns) as row length in halfwords, and use double precision store.

MR0SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR0SNP Size of Code Area: 16 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MR0SNP

Function: Moves a scalar value to all positions of a partition of a REMOTE single precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$M_{A\ TO\ B,C\ TO\ D} = X;$ where X is a scalar, and M is a REMOTE single precision matrix.

☐ Other Library Modules:

Execution Time (microseconds): 22.8 + n(5.6 + 8.6m) for an n x m partition.

Input Arguments:

Type	Precision	How Passed	Units
scalar	SP	F0	-
integer(n)	SP	R5	-
integer(m)	SP	R6	-
integer(outdel)	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
matrix(n,m)	SP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm: Find row length in halfwords by SLL # columns, 1 add row length to outdel

```
Loop:  Indexing on # rows, using BCTB
   Loop:  Indexing on # columns, using BCTB
          Store scalar in pointed to output element
   End.
   Add outdel (with row size) to output pointer
End.
```


MR1DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR1DNP Size of Code Area: 22 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MR1DNP

Function: Moves a partition of a double precision matrix to a partition of a double precision matrix. At least one of the matrices has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$M1 = M2_{A\ TO\ B,C\ TO\ D};$ where M1 and M2 are double precision matrices,
$M1_{A\ TO\ B,C\ TO\ D} = M2;$ and at least one of M1 and M2 is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 28.4 + n(8.2 + 15.0m) for nxm partition.

Input Arguments:

Type	Precision	How Passed	Units
matrix(n,m)	DP	R4 → ZCON → 0th element	-
integer(rows)	SP	R5	-
integer(columns)	SP	R6	-
integer(indel,outdel)	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
matrix(n,m)	DP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:
Same as MR1SNP, except use double precision loads and stores and use 4 · (# columns) as row length.

MR1SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR1SNP **Size of Code Area:** 22 Hw

Stack Requirement: 22 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MR1SNP

Function: Moves a partition of a single precision matrix to a partition of a single precision matrix. Either or both matrices have the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$M1 = M2_{A\ TO\ B,C\ TO\ D}$; where M1 and M2 are single precision matrices,
$M1_{A\ TO\ B,C\ TO\ D} = M2$; and at least one of M1, M2 is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 28.4 + n(8.2 + 12.6m) for nxm partition.

Input Arguments:

Type	Precision	How Passed	Units
matrix(n,m)	SP	R4 → ZCON → 0th element	-
integer(n)	SP	R5	-
integer(m)	SP	R6	-
integer(indel,outdel)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
matrix (n,m)	SP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm: Separate indel and outdel into separate registers. Find row size, in halfwords, of result matrix by shifting left 1, # columns
Add rowsize to indel
Add rowsize to outdel
Loop: Indexing on # of rows of output, and using BCTB
Loop: Indexing on # of columns of input, using BCTB
load (single precision) pointed to input element
store (single precision) pointed to output element
End.
Add indel (with row size added) to input pointer
Add outdel (with row size added) to output pointer
End.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

MR1TNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR1TNP **Size of Code Area:** 24 Hw

Stack Requirement: 22 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MR1TNP

Function: Moves a partition of a double precision matrix to a partition of a single precision matrix. At least one of the matrices has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$M1 = M2_{A\ TO\ B,C\ TO\ D}$; where M1 is a single precision matrix, M2 is a
$M1_{A\ TO\ B,C\ TO\ D} = M2$; double precision matrix, and at least one of
M1 and M2 is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 31.2 + n(7.6 + 13.8m) for nxm partition.

Input Arguments:

Type	Precision	How Passed	Units
matrix(n,m)	DP	R4 → ZCON → 0th element	-
integer(rows)	SP	R5	-
integer(columns)	SP	R6	-
integer(indel,outdel)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
matrix (n,m)	SP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:
Same as MR1SNP, except use double precision load for index alignment, and use 4 · (# columns) as the length in halfwords of double precision partition.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

MR1WNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR1WNP **Size of Code Area:** 24 Hw

Stack Requirement: 22 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MR1WNP

Function: Moves a partition of a single precision matrix to a partition of a double precision matrix. Either or both matrices have REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

M1 = M2 a TO b, c TO d; where M1 is a double precision matrix, and M2 is a single precision matrix, and

M1 a TO b,c TO d = M2; at least one of M1 or M2 is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 32.8 + n(8.2 + 13.8m) for nxm partition.

Input Arguments:

Type	Precision	How Passed	Units
matrix	SP	R4 → ZCON → 0th element	-
integer(rows)	SP	R5	-
integer(columns)	SP	R6	-
integer(indel,outdel)	(SP,SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
matrix	DP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as for MR1SNP, except use double precision stores after zeroing the low half of the floating point register.

VR0DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR0DN **Size of Code Area:** 6 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR0DN

Function: Moves a scalar to all elements of a double precision vector with the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V}$ = 0; V a REMOTE double precision vector.

☐ Other Library Modules:

Execution Time (microseconds): 16.4 + 9.2n, n = size of vector.

Input Arguments:

Type	Precision	How Passed	Units
scalar	DP	F0	-
integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
vector(n)	DP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR0SN, except use double precision store.

VR0DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR0DNP Size of Code Area: 10 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR0DNP

Function: Moves a scalar to all elements of a column of a double precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$M_{*,I} = 0$; M a double precision REMOTE matrix.

☐ Other Library Modules:

Execution Time (microseconds): 21.2 + 10.0n, n = length of vector result.

Input Arguments:

Type	Precision	How Passed	Units
scalar	DP	F0	-
integer (n)	SP	R5	-
integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
vector(n)	DP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm: Same as VR0SNP, except use double precision stores.

VR0SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR0SN Size of Code Area: 6 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR0SN

Function: Moves a scalar to all elements of a single precision vector with the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$\overline{V} = 0$; V a REMOTE single precision vector.

☐ Other Library Modules:

Execution Time (microseconds): $16.4 + n \cdot 8.0$, n = size of vector.

Input Arguments:

Type	Precision	How Passed	Units
scalar	SP	F0	-
integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
vector(n)	SP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments: Registers Unsafe Across Call: F0,F1.

Algorithm:
Store elements in reverse order using the input length both as an index and to control the loop.

VR0SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR0SNP **Size of Code Area:** 10 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: none.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR0SNP

Function: Moves a scalar to all elements of a column of a single precision matrix.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$M_{*,I} = 0$; M a single precision REMOTE matrix.

☐ Other Library Modules:

Execution Time (microseconds): 21.2 + 8.8n, n = length of vector result.

Input Arguments:

Type	Precision	How Passed	Units
scalar	SP	F0	-
integer(n)	SP	R5	-
integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
vector(n)	SP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Store elements one at a time, adding outdel to the pointer after each store.

VR1DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1DN **Size of Code Area:** 8 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR1DN

Function: Moves a double precision vector to a double precision vector, where at least one of the vectors has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V2 = V1; where V1 or V2 has been declared a REMOTE vector, and V1, V2 are both double precision.

☐ Other Library Modules:

Execution Time (microseconds): 16.4 + n · 15.0, n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
vector(n)	DP	R4 → ZCON → 0th element	-
integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
vector(n)	DP	R1 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SN, except use double precision loads and stores.

VR1DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1DNP Size of Code Area: 20 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR1DNP

Function: Moves a double precision vector to a double precision vector when elements of source or receiver are not contiguous, and at least one has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$V = M_{*,J}$; where V is declared double precision vector, M is double precision
$M_{*,J} = V$; matrix, and V or M is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 17.0n + 29.6 if neither input nor output is contiguous. 17.0n + 30.4 if either input or output is contiguous, where n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
vector (n)	DP	R4 → ZCON → 0th element	-
integer (n)	SP	R5	-
integer(indel)	SP	R6	-
integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
vector (n)	DP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SNP, except if indel or outdel = 0, sets to 4, and does double precision loads and stores.

VR1SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1SN Size of Code Area: 8 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR1SN

Function: Moves a single precision vector to a single precision vector, where at least one of the vectors has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

V1 = V2; where V1 or V2 or both are remote and V1 and V2 are single precision.

☐ Other Library Modules:

Execution Time (microseconds): 16.4 + 12.6n, n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
vector(n)	SP	R4→ ZCON → 0th element	-
integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
vector(n)	SP	R2→ ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Loops n times, using length both as index and to control the loop. Load, then store, each element in turn.

VR1SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1SNP Size of Code Area: 20 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR1SNP

Function: Moves a single precision vector to a single precision vector when elements of source or receiver are not contiguous and at least one has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

V = M*,J; where V is declared single precision vector, M is a
M*,J = V; single precision matrix, and V or M has the REMOTE attribute.

☐ Other Library Modules:

Execution Time (microseconds): 14.6n + 30.4 if either input or output is contiguous. 14.6n + 29.6 if neither input nor output is contiguous, where n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
vector(n)	SP	R4 → ZCON → 0th element	-
integer (n)	SP	R5	-
integer(indel)	SP	R6	-
integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
vector	SP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

If outdel = 0, sets it to 2.
If indel = 0, sets it to 2.

Loops 'length' times, moving one element each loop. Adds indel to input pointer and outdel to output pointer after each move.

VR1TN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1TN Size of Code Area: 8 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR1TN

Function: Moves a double precision vector to a single precision vector, where at least one of the vectors has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

V1 = V2; where V1 is a single precision vector, V2 is a double precision vector, and at least one of V1 and V2 is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 16.4 + 13.8n, n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
vector (n)	DP	R4 → ZCON → 0th element	-
integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
vector	SP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SN, except use double precision loads.

VR1TNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1TNP **Size of Code Area:** 20 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR1TNP

Function: Moves a double precision vector to a single precision vector, when elements of source or receiver are not contiguous, and at least one of them has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$V = M_{*,J}$; where V is a single precision vector, M is a double precision matrix, and V or M is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 15.8n + 30.4 if either input or output is contiguous.
15.8n + 29.6 if neither input nor output is contiguous, where n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
vector(n)	DP	R4 → ZCON → 0th element	-
integer(n)	SP	R5	-
integer(indel)	SP	R6	-
integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
vector	SP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SNP except if indel = 0, sets it to 4, and does double precision loads.

VR1WN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1WN **Size of Code Area:** 10 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR1WN

Function: Moves a single precision vector to a double precision vector, where at least one of the vectors has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:
V1 = V2; V1 or V2 remote, V1 double precision, and V2 single precision.

☐ Other Library Modules:

Execution Time (microseconds): 20.6 + 13.8n, n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
vector(n)	SP	R4 → ZCON → 0th element	-
integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
vector(n)	DP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SN, except use double precision store with low half of floating register zeroed.

VR1WNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1WNP **Size of Code Area:** 22 Hw

Stack Requirement: 18 Hw **Data CSECT Size:** 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VR1WNP

Function: Moves a single precision vector to a double precision vector, when elements of source or receiver are not contiguous, and at least one of them has the REMOTE attribute.

Invoked by:

☒ Compiler emitted code for HAL/S construct of the form:

$V = M_{*,J}$; where V is a double precision vector, M is a single precision matrix, and V or M is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 15.8n + 31.2 if either input or output is contiguous.
15.8n + 32.0 if neither input nor output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
vector(n)	SP	R4 → ZCON → 0th element	-
integer(n)	SP	R5	-
integer(indel)	SP	R6	-
integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
vector(n)	DP	R2 → ZCON → 0th element	-

Errors Detected:

Error #	Cause	Fixup

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SNP, except if outdel = 0, sets it to 4, and uses double precision stores, after clearing the low half of the floating point register.

6.0 SYSTEM INTERFACES

This section deals with characteristics and behavior of the HAL/S-FC compiler as related to the environment in which the compiler operates. Specifically, these items are in relation to the host computer in which the compiler is executed.

6.1 Internal System Interfaces

The HAL/S-FC compiler is designed to operate under OS/360 MVT or an equivalent operating system (such as OS/VS2 on IBM 370 equipment.) The compiler was developed under Release 21.6 of OS and uses many of the features of that system.

6.1.1 Macro Instructions

All operating system communication is performed via standard assembler language macro instructions as provided with OS MVT. The following list contains the names of all macros executed directly by the HAL/S-FC compiler.

ABEND	BLDL	CHECK	CLOSE	DCB
DCBD	DELETE	FIND	FREEMAIN	FREEPOOL
GET	GETBUF	GETMAIN	LOAD	NOTE
OPEN	POINT	PUT	READ	RETURN
SAVE	SPIE	STIMER	STOW	TIME
TTIMER	WRITE			

The forms of some of these macros require further explanation:

FREEMAIN - All FREEMAIN macros use the SP parameter to indicate subpool 22. Both freeing of single elements of storage and freeing of an entire subpool are performed.

GETMAIN - All requests for main storage are made with the SP operand specifying subpool 22. GETMAINs are done for both single elements of storage of specific size and once, during compiler initialization, for a variable region using the VC form of GETMAIN. This initialization GETMAIN obtains the largest contiguous element of memory available in the job step region. This memory (assigned to subpool 22) is used to hold executable compiler code and as a data area for the compiler.

STIMER - The STIMER macro with the TASK option is used to start an accounting of CPU time used by the compiler.

TTIMER - The TTIMER macro is used to test the TASK interval timer as started by the STIMER macro to determine elapsed CPU time at various points in a compilation.

6.1.2 Dynamic Invocation of the Compiler

The HAL/S-FC compiler may be dynamically invoked by another processing program. The details of this interface are controlled by the HAL/SDL Interface Control Document.

The dynamic invocation capability allows:

- specification of a parameter string to be acted upon by the compiler,
- specification of an alternate DDNAME list for those DD cards referenced by the compiler, and
- specification of communication areas in which the compiler will supply information to the invoking program.

The compiler takes the following actions to restore its environment upon return to the program which performed the invocation.

- All DCB's opened by the compiler are closed and any automatically acquired buffers are FREEPOOLed.
- All GETMAINed storage is FREEMAINed.
- The SPIE exit (if any) is restored to its status upon entering the compiler.

6.1.3 OS/360 Access Methods

In performing input/output processing the HAL/S-FC compiler uses the OS Data Management Access Methods:

BSAM QSAM BPAM

No other access methods are used, and all datasets manipulated by the compiler are standard OS/360 datasets.

6.2 User or External System Interfaces

The majority of ways in which users of the HAL/S-FC compiler interact with the compiler are described in Sections 2 through 5 of this document. However, the primary vehicle for user communication with this system is Job Control Language which is a part of the compiler's interface to the system in which it operates. This subsection describes the two areas of external or user interfaces to the system:

1) user-defined options acted upon by the compiler, and
2) the JCL with which the user defines the compiler's data and hence the environment in which the compiler is to operate.

6.2.1 User-defined Options

The HAL/S-FC compiler has a number of optional features which may be exercised by the user. These options are indicated via keyword parameters passed to the compiler in the standard OS/360 method. The options are either passed to the compiler during dynamic invocation as described in the HAL/SDL ICD, or are passed via the PARM field on the EXEC card in the JCL invoking the compiler. A list of these options and their effects may be found in Appendix A.

6.2.2 Job Control Language Specification

JCL is the means by which any user of the compiler defines the set of data upon which the compiler is to operate. This JCL is therefore the first interface of the user and the compiler. Once this set of data is specified, all other interfaces with the user are through this data in the manner described in preceeding chapters. The remainder of this subsection consists of two parts: 1) a listing of some typical JCL for compiler invocation; and 2) a chart describing the uses, presumed attributes, and access methods for all DD cards.

```
//HALFC    PROC OPTION='LEVEL=HALS101                                    00010000
//HAL      EXEC PGM=MONITOR,REGION=350K,TIME=1,                          00020000
//             PARM='NOZCON,&OPTION'                                     00030000
//STEPLIB  DD DISP=SHR,DSN=&LEVEL..MONITOR                               00040000
//PROGRAM  DD DISP=SHR,DSN=&LEVEL..COMPILER                              00050000
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)            00060000
//LISTING2 DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)            00070000
//OUTPUT3  DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),              00080000
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),                       00090000
//            DSN=&&HALOBJ                                               00100000
//OUTPUT4  DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)               00110000
//OUTPUT5  DD DISP=(MOD,PASS),DSN=&&HALSDF,SPACE=(TRK,(2,2,1)),          00110000
//            DCB=(RECFM=F,LRECL=1680,BLKSIZE=1680),UNIT=SYSDA           00120000
//OUTPUT6  DD DISP=(MOD,PASS),DSN=&&TEMPLIB,SPACE=(TRK,(2,2,1)),         00130000
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),UNIT=SYSDA            00140000
//OUTPUT7  DD DUMMY,DCB=(RECFM=FM,LRECL=121,BLKSIZE=1680),UNIT=SYSDA      00150000
//ERROR    DD DISP=SHR,DSN=&LEVEL..ERRORLIB                              00160000
//FILE1    DD UNIT=SYSDA,SPACE=(CYL,3)                                   00170000
//FILE2    DD UNIT=SYSDA,SPACE=(CYL,3)                                   00180000
//FILE3    DD UNIT=SYSDA,SPACE=(CYL,3)                                   00190000
//FILE4    DD UNIT=SYSDA,SPACE=(CYL,3)                                   00200000
//FILE5    DD UNIT=SYSDA,SPACE=(CYL,3)                                   00205000
//FILE6    DD UNIT=SYSDA,SPACE=(CYL,3)                                   00210000
                                                                         00220000
```

Typical JCL for Compiler Invocation

Compiler DDNAMES, Uses, and Requirements

DDNAME	FUNCTION	DEVICE REQUIREMENTS	LRECL	RECFM	BLKSIZE	BUFNO[3]	DSORG	ACCESS METHOD, MACRF
PROGRAM	executable compiler phases	· direct access · magnetic tape	7200	F	7200	0	PS	BSAM, R
SYSPRINT	Primary listing	· printer · intermediate storage	133	FBA	3458[1]	1	PS	QSAM, PL
LISTING2	Secondary unformatted listing	· printer · intermediate storage	133	FBA	3458[1]	1	PS	QSAM, PL
OUTPUT3	object module output	· direct access · magnetic tape · card punch	80	FB	400[1]	1	PS	QSAM, PL
OUTPUT4	duplicate object module output	· direct access · magnetic tape · card punch	80	FB	400[1]	1	PS	QSAM, PL
OUTPUT5	Simulation data file output	· direct access	1680[4]	F[4]	1680[4]	0	PO	BPAM, W
OUTPUT6	Template search and creation	· direct access	80[2]	FB	1680[2]	1	PO	BPAM, WR
OUTPUT7	pseudo-assembly listing for link-edit ABSLIST functio	· direct access · magnetic tape	133	FBM	3458[1]	1	PS	BSAM, PL
ERROR	Compiler error message retrieval	· direct access	80	FB	400	1	PO	BPAM, R
FILE1	Phase 1, Optimizer HALMAT work file Auxiliary HALMAT file	· direct access	7200	F	7200	0	PS	BSAM, RWP

Compiler DDNAMES, Uses, and Requirements (Con't)

DDNAME	FUNCTION	DEVICE REQUIREMENTS	LRECL	RECFM	BLKSIZE	BUFNO[3]	DSORG	ACCESS METHOD, MACRF
FILE2	Literal communication area	· direct access	1560	F	1560	0	PS	BSAM, RWP
FILE3	Phase I Init/Const work area Phase II code gen. work area	· direct access	1600	F	1600	0	PS	BSAM, RWP
FILE4	Phase 2 HALMAT work file	· direct access	7200	F	7200	0	PS	BSAM, RWP
FILE5	Phase III paging area	· direct access	1680	F	1680	0	PS	BSAM, RWP
FILE6	Statement data communication area	· direct access	512	F	512	0	PS	BSAM, RWP
SYSIN	Primary source input	· card reader · intermediate storage	80 <= LRECL <= 132	FB	legal multiple of LRECL[1]	1	PS	QSAM, GL
INCLUDE	Secondary source input	· direct access	80 <= LRECL <= 132	FB	legal multiple of LRECL[1]	1	PO	BPAM, R
ACCESS	ACCESS Rights control	· direct access	80[2]	FB	1680[2]	1	PO	BPAM, R

Notes:

1. BLKSIZE value may be altered by user to any installation-legal value.
2. Compiler will use LRECL and BLKSIZE supplied by user.
3. BUFNO may be specified by user for any PS type datasets.
4. Defaults are shown; Records are always written as 1680 blocks but user-supplied attributes will be retained.

Appendix A

Compile-Time JCL Options

This Appendix describes the compiler options which may be coded in the PARM field of the EXEC card in the Job Control Language invoking the compiler. In all cases, options are separated in the PARM field by commas. If an option is referenced more than once in a PARM field, the last reference (scanning left to right) will be used to determine the option's setting.

There are two general classes of options recognized by the compiler: Type 1 options having a binary value of "on" or "off", and Type 2 options having a numeric or string value.

Type 1 Options

Type 1 options are controlled by keywords in the PARM field. The appearance of the keyword indicates that the option is to be "on" during the compilation unless the keyword is preceded by the characters "NO" in which case the option is "off". Some Type 1 options have alternate, shorter spellings which may be used interchangably with the standard keywords.

When a Type 1 option has an alternate form, the negative or "off" value (equivalent to adding 'NO' to the standard keyword) is specified by preceeding the alternate form with the character 'N'. The 'NO' and 'N' notations may only be used with the standard and alternate forms respectively. For example, the LIST option has the alternate form L. If the negative is to be specified, it may be done as NOLIST or NL; NLIST or NOL will not be recognized.

The following Type 1 options are recognized. The default settings shown are used in the absence overriding PARM field specifications.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Keyword	Alternate	Default	Function
LISTING2	L2	off	Causes unformatted source listing to be generated.
DUMP	DP	off	Requests the compiler to produce a memory dump if certain internal compiler errors occur.
LIST	L	off	Produces an assembly listing from Phase II of the compiler.
TRACE	TR	on	Causes the generation of a link to the HSS end-of-statement routine in the object module. Enables Real Time execution and debugging.
DECK	D	off	Controls production of an additional object deck on the OUTPUT4 DD card.
TABLST	TL	off	Causes Phase III of the compiler to produce formatted dump of the simulation data file (SDF).
SRN	none	off	Causes the compiler to omit the last eight columns or characters from the source scanning. These columns are then used to print information on the listing.
TABLES	TBL	on	Controls generation of Simulation Data Files.
ADDRS	A	off	Indicates the presence of statement address information in the Simulation Data Files.
ZCON	Z	on	Indicates external linkage conventions to be used (via Z_CONs or direct).
TABDMP	TBD	off	Causes Phase III of the compiler to produce a hexadecimal dump of the simulation data file.

Keyword	Alternate	Default	Function
SDL	none	off	Informs the compiler that it is operating within the SDL. ACTIONS specific to SDL operations are keyed to this option such as inclusion of SRN, Change Authorization Field and Source record revision indicator on primary listing.
FCDATA	FD	off	Causes HAL/S-360 data to be allocated using a halfword as the basic memory unit. This causes data area allocation which maps directly into HAL/S-FC data allocation.
ZCON	Z	on	Causes calls to out-of-line routines to be performed via long indirect address constants (ZCONs).
SCAL	SC	on	Allows use of SCAL/SRET instructions for subroutine linkage. If off, BAL linkage is used. SCAL is meaningless if NOMICROCODE is specified.
MICROCODE	MC	on	Allows use of instructions which only exist on late versions of the Space Shuttle GPC. This includes SCAL, SRET, MVS, MVH and BIX. Use of SCAL and SRET may be separately controlled with the SCAL option.
SREF	SR	off	Causes special processing of user-defined symbols which appear within an EXTERNAL COMPOOL template which is included in another compilation. Any items in such a COMPOOL which are not referenced by the primary compilation unit are not printed in the symbol table listing.

Type 2 Options

Type 2 options have "values" which may be altered by the user. The values are specified by including the pseudo-assignment statement:

...,<type 2 opt>=<value>,...

in the PARM field where <type 2 opt> is one of the legal type 2 options, and <value> is the value to be used during compiler execution. The form of <value> is determined by the specific options. Some Type 2 options have alternate, shorter spellings which may be used interchangably with the standard forms.

The following Type 2 options are recognized. The default values shown are used in the absence of overriding PARM field specifications.

Standard	Alternate	Default	Function
PAGES=	P=	250	Sets the maximum page number to be allowed in generation of the primary compilation listing.
LINECT=	LC=	59	Sets the maximum number of lines which will be printed on any one page of either the primary or secondary source listing.
TITLE=	T=	null	Specifies 1 to 60 characters used by the compiler when printing header information at the top of each page of the listing.
SYMBOLS=	SYM=	200	Specifies the size of the compiler's symbol table.
MACROSIZE=	MS=	500	Specifies the maximum number of characters allowed in text of macro definitions.

Standard	Alternate	Default	Function
LITSTRINGS=	LITS=	2000	Specifies the maximum total number of characters allowed in character literals in a table.
COMPUNIT=	CU=	0	Specifies a compilation unit number to identify the unit of compilation. The number is made available in the SDF and in the Block Data Areas for code blocks in a HAL/S-FC compilation.
XREFSIZE=	XS=	2000	Specifies the number of cross reference table entries allocated by the compiler. Each entry used 4 bytes of storage.
CARDTYPE=	CT=	null	Specifies pairs of characters which define a mapping of arbitrary input record types (column 1 of the record) into the standard types (E,M,S,C, D, and blank). E.g. CT=XYCM would cause any 'X' records to be compiled as comments and any 'T' records to be compiled as 'M' records.
LABELSIZE=	LBLS=	1200	Specifies the maximum number of internal label points which will be maintained by the code generator.
DSR=	none	1	Specifies the value to be used for the data sector register in the right hand halfword of the R2 operand of the MVH instruction. The compiler will use this value explicitly when it is not possible to use a standard Z-type address constant in which the DSR field is filled in by the linkage editor.
BLOCKSUM=	BS=	400	Specifies the size of the table used for collecting information for printing BLOCK SUMMARY information.

Appendix B

Compiler Directives

The following compiler directives have been defined for the HAL/S-FC compiler.

a) The DEVICE directive has the form:

```
D DEVICE CHANNEL=n <option>
```

This option is accepted by the HAL/S-FC compiler to insure compatability of source input with the HAL/S-360 compiler. It has no effect in HAL/S-FC operation.

b) The INCLUDE directive has the form:

```
D INCLUDE <name> <option>
```

or

```
D INCLUDE TEMPLATE <unit name> <option>
```

This directive names a member of an include library as defined in Section 2.2. The <option> may be "NOLIST" or null. The "NOLIST" option indicates that the included text is not to be listed.

c) The PROGRAM directive has the form:

```
D PROGRAM ID=<id>
```

This directive provides a Program Identification Name to be used by the compiler to determine access rights to controlled resources as described in Section 2.3.

d) The version directive has the form:

```
D VERSION @
```

This directive provides version information for templates. It is generated and checked automatically by the compiler.

e) The EJECT directive has the form:

```
D EJECT
```

and causes the compiler to eject a page before resuming generation of the primary source listing. The EJECT directive record is not printed on the listing.

f) The SPACE directive has the form:

```
D SPACE[<n>]
```

where <n> is an integral number.

This directive causes the compiler to skip <n> lines prior to generation of the next line in the primary source listing. The <n> may be omitted in which case one line is skipped. If <n> is greater than 3, 3 lines are skipped. The SPACE directive is not printed on the listing.

Appendix C

Error Classifications

Note: "b" denotes a blank.

CLASS A: ASSIGNMENT STATEMENTS

A	ARRAY ASSIGNMENT
V	COMPLEX VARIABLE ASSIGNMENT
b	MISCELLANEOUS ASSIGNMENT

CLASS B: COMPILER TERMINATION

B	HALMAT BLOCK SIZE
N	NAME SCOPE NESTING
S	STACK SIZE LIMITATIONS
T	TABLE SIZE LIMITATIONS
X	COMPILER ERRORS
b	MISCELLANEOUS

CLASS C: COMPARISONS

b	GENERAL COMPARISONS

CLASS D: DECLARATION ERRORS

A	ATTRIBUTE LIST
C	STORAGE CLASS ATTRIBUTE
D	DIMENSION
F	FUNCTION DECLARATION
I	INITIALIZATION
L	LOCKING ATTRIBUTE
Q	STRUCTURE TEMPLATE TREE ORGANIZATION
S	FACTORED/UNFACTORED SPECIFICATION
T	TYPE SPECIFICATION

```
   U        UNDECLARED DATA
   b        MISCELLANEOUS


CLASS E:  EXPRESSIONS

   A        ARRAYNESS
   B        BIT STRING EXPRESSIONS
   C        CROSS PRODUCT
   D        DOT PRODUCT
   L        LIST EXPRESSIONS
   M        MATRIX EXPRESSIONS
   O        OUTER PRODUCT
   V        VECTOR EXPRESSIONS
   b        MISCELLANEOUS EXPRESSIONS


CLASS F:  FORMAL PARAMETERS & ARGUMENTS

   D        DIMENSION AGREEMENT
   N        NUMBER OF ARGUMENTS
   S        SUBBIT ARGUMENTS
   T        TYPE AGREEMENT


CLASS G:  STATEMENT GROUPINGS (DO GROUPS)

   B        BIT TYPE CONTROL EXPRESSION
   C        CONTROL EXPRESSION
   E        EXIT/REPEAT STATEMENTS
   L        END LABEL
   V        CONTROL VARIABLE


CLASS I:  IDENTIFIERS

   L        LENGTH
   R        REPLACED IDENTIFIERS
   S        QUALIFIED STRUCTURE NAMES


CLASS L:  LITERALS

   B        BIT STRING
   C        CONVERSION TO INTERNAL FORMS
   F        FORMAT OF ARITHMETIC LITERALS
   S        CHARACTER STRING


CLASS M:  MULTILINE FORMAT

   C        OVERPUNCH CONTEXT
   E        E-LINE
   O        OVERPUNCH USE
```



```
   S        S-LINE
   b        COMMENTS


CLASS P:  PROGRAM CONTROL & INTERNAL CONSISTANCE

   A        ACCESS CONTROL
   C        COMPOOL BLOCKS
   D        DATA DEFINITION
   E        EXTERNAL TEMPLATES
   F        FUNCTION RETURN EXPRESSIONS
   L        LABELS
   M        MULTIPLE DEFINITIONS
   P        BLOCK DEFINITION
   S        PROCEDURE/FUNCTION TEMPLATES
   T        TASK DEFINITIONS
   U        CALLS FROM UPDATE BLOCKS
   b        MISCELLANEOUS


CLASS Q:  SHAPING FUNCTIONS

   A        ARRAYNESS
   D        DIMENSION INFORMATION
   S        SUBSCRIPTS
   X        ARGUMENT TYPE


CLASS R:  REAL TIME STATEMENTS

   E        ON/SEND ERROR STATEMENTS
   T        TIMING EXPRESSIONS
   U        UPDATE BLOCKS


CLASS S:  SUBSCRIPT USAGE

   C        SUBSCRIPT COUNT
   P        PUNCTUATION
   Q        PRECISION QUALIFIER
   R        RANGE OF SUBSCRIPT VALUES
   S        USAGE OF ASTERISKS
   T        SUBSCRIPT TYPE
   V        VALIDITY OF USAGE


CLASS T:  I/O STATEMENTS

   C        CONTROL
   D        DEVICE NUMBER
   b        MISCELLANEOUS


CLASS U:  UPDATE BLOCKS
```

I	IDENTIFIER USAGE
P	PROGRAM BLOCKS
T	I/O

CLASS V: COMPILE-TIME EVALUATIONS

A	ARITHMETIC OPERATIONS
C	CATENATION OPERATIONS
E	UNCOMPUTABLE EXPRESSIONS
F	FUNCTION EVALUATION

CLASS X: IMPLEMENTATION DEPENDENT FEATURES

A	PROGRAM ID DIRECTIVE
D	DEVICE DIRECTIVE
I	INCLUDE DIRECTIVE
U	UNKNOWN OR INVALID DIRECTIVE

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840